Full modification coverage through automatic similarity-based test case selection

Francisco G. de Oliveira Neto\textsuperscript{a}, Patrícia D. L. Machado\textsuperscript{a}, Richard Torkar\textsuperscript{b,c,d}

\textsuperscript{a}Federal University of Campina Grande, PB, Brazil  
\textsuperscript{b}Chalmers and the University of Gothenburg, Sweden  
\textsuperscript{c}Mälardalen University, Sweden  
\textsuperscript{d}Huawei Technologies Sweden AB, Sweden

Abstract

Context: This paper presents the similarity approach for regression testing (SART), where a similarity-based test case selection technique is used in a model-based testing process to provide selection of test cases exercising modified parts of a specification model. Unlike other model-based regression testing techniques, SART relies on similarity analysis among test cases to identify modifications, instead of comparing models, hence reducing the dependency on specific types of model.

Objective: To present convincing evidence of SART’s effectiveness.

Method: We investigate SART in a case study and an experiment. The case study uses artifacts from industry and should be seen as a sanity check of SART, while the experiment focuses on gaining statistical power through the generation of synthetical models in order to provide convincing evidence of SART’s effectiveness. Through posthoc analysis we obtain p-values and effect sizes to observe statistically significant differences between treatments with respect to transition and modification coverage.

Results: The case study with industrial artifacts revealed that SART is able to uncover the same number of defects as known similarity-based test case selection techniques. In turn, the experiment shows that SART, unlike the other investigated techniques, presents 100% modification coverage. In addition, all techniques covered a similar percentage of model transitions.

Conclusions: In summary, not only does SART provide transition and defect coverage equal to known STCS techniques, but it exceeds greatly in covering modified parts of the specification model, being a suitable candidate for model-based regression testing.

Keywords: Regression testing, Test case selection, Model-based testing

1. Introduction

Regression testing at code level has been widely investigated in literature [1] and enables solution for most software’s structural testing problems such as modified code coverage or finding data and control dependencies affected by modifications. At the same time, there has been a growing interest in model-based regression testing due to the many benefits of handling high-level abstraction models [2, 3]. Furthermore, existing model-based testing (MBT) techniques enable automatic generation of test cases from models, making it easier to design and execute test cases. On the other hand, those generated test suites, very often, include numerous redundant test cases that can make it significantly more difficult to execute regression tests [4].

Rather then executing all test cases (i.e. the retest all approach), a more feasible approach to regression testing is to select a few test cases to execute. But even with a specification model, particularly a high level one (e.g. activity diagrams, or natural language use case templates) where visualisation and readability assists testers in understanding the system under test (SUT), selecting test cases can be costly and overwhelming since testers need to be aware of a SUT’s new, obsolete and unchanged elements.

1.1. Problem Statement

Given the amount of high level information available, the main challenge then becomes about selecting a representative subset of test cases in order to reduce the costs of regression testing at the system level. In other words, we aim at maximizing the chances of detecting defects as well as minimizing the number of test cases needed. Here, we consider as representative the test cases exercising modifications performed on a software system—a very common criterion for selecting regression test cases [3, 5]. But how can we then identify these test cases in a test suite?

Existing selection techniques answer that question by using different approaches. Either by comparing different versions of specification models, or analysing existing dependencies between model elements. As an example, consider the state-based specification models presented in Figure 1 and the correspondent test suites that can be generated from them, by traversing all paths. They represent the specification for beginning a game. Note that Model 2 has two modifications: removal of state 4 and then addition of state 8\textsuperscript{1}. If we could not execute all test cases, then a tester would prefer to execute only TC’2 to verify whether this modification affects the proper execution of States

\textsuperscript{1}For each modified state, their connecting transitions were, respectively, removed and added.
1, 3, 6 and 7. If another test case could be selected, a good option would be TC'3, since State 5 is closer\(^2\) to the modification performed.

Determining modifications by simply comparing those models can be misleading, inefficient and at the same time non-trivial. First, the technique needs to be aware of both the model layout and the model elements (states, labels, conditions, etc.) Otherwise, modifications that do not change the model layout may not be detected. In addition, a straightforward comparison can erroneously determine that only the labels of transitions/states are changed when, in fact, a completely new location in the execution scenario is added. Note that, in order to select the test cases, the technique also requires traceability between model elements and the steps of the test cases.

Certainly, for this example, applying a comparison technique seems like an effortless task. However, for large and complex models, comparison and analysis of all model elements can be inefficient, specially when specifications become large or unstructured, after consecutive modifications. Similarly, dependency analysis is costly because the number of dependencies can grow significantly at each modification. Furthermore, requiring traceability between model elements and test cases introduces a level of dependency to the type of model being used. Being dependent to a specific type of model is risky given that it can compromise the versatility of a technique, whenever stakeholders decide to represent their specification according to different types of models.

### 1.2. Proposed Solution

In turn, similarity-based test case selection (STCS) relies on similarity/distance functions to select the more (or less) different scenarios, hence enabling removal of redundancy among test cases [6, 7]. The benefit with this type of selection is testing a diversity of test cases in a SUT. Besides, similarity functions are usually mathematical functions easy to define and incorporate in a tool. They have now been widely used and investigated regarding their capability to identify similarities within a set of test cases [7].

In the scope of model-based regression testing, our contribution is then to use those similarity measurements to assess sets of test cases belonging to different software versions and then use this information to select test cases for system testing of a modified specification. We named our proposed technique the similarity approach for regression testing, or simply SART. Different from other specification-based approaches presented in literature [8], we focus on similarity analysis among test cases to identify modifications instead of comparing and analysing models. Consequently, SART is not dependent to a specific type of model. Considering our example in Figure 1, our technique would simply analyse the test cases’ content by converting them into vectors. By looking at the distance between those vectors SART is able to determine the test cases exercising modifications. Ultimately, SART would select TC'2 and TC'3 to test both modifications performed.

---

\(^2\)State 3 was directly affected by the addition of State 8 because a new transition was added to it.
In order to gather evidence regarding SART’s feasibility, we performed a case study and an experiment. First, we analysed SART with industrial artifacts that include a specification models, a SUT and the detected defects\(^3\). Complementary to our case study, we also performed an experiment where SART selects test cases automatically generated from a larger sample of synthetic models obtained through stochastic generation [9]. We compare SART with other STCS techniques, and both analyses revealed promising results regarding SART’s capabilities, and some drawbacks limiting our selection algorithm. For instance, even though a high modification coverage is achieved, SART shows no significant improvement regarding transition coverage with respect to the compared techniques. At the same time, all investigated techniques revealed the same number of faults in our case study, indicating that SART manages to provide similarity-based test case selection, but unlike existing STCS techniques, it targets selection of modification-traversing test cases.

The remainder of our paper is structured as follows. Section 2 presents the MBT concepts and the type of model used in our MBT approach, whereas Section 3 presents a background on specification-based regression testing. Then we present details on SART’s selection strategy (Section 4), followed by Section 5 describing our empirical evaluation. Related work is discussed in Section 6 and, finally, we draw conclusions and discuss future work in Section 7.

2. Model-based testing

Model-based regression testing relies on model-based testing (MBT) approaches to enable the use of software models in order to automate generation/selection of test cases. The goal is to use techniques based on different coverage criteria to systematically harness information from models (code or specification levels) and then generate, manage or even execute test cases. Thus, tools can be developed to automatically analyse and explore the SUT and discover intrinsic testing scenarios that, otherwise, would have been hard to identify. On the other hand, much of those testing scenarios can be redundant or even irrelevant for regression testing.

Our work focuses on test cases describing software system behaviour, thus our test cases are usually described in natural language and need to be executed manually (i.e. abstract test cases). Albeit there is a difficulty in tracing abstract test cases to the respective executable code parts it is, however, easier to identify which functionalities or use case scenarios are being tested. The specification model used in the case study presented in this paper is a labeled transition system (LTS) model that can be obtained, for instance, from use case documents to provide an intermediary model format for automatic test case generation (an example of LTS is shown Figure 1) [10]. An LTS is defined as a 4-tuple \( S = (Q; A; T_r; q_0) \), where: \( Q \) is the set of states, \( q_0 \in Q \) being the initial state; \( A \) is a finite non-empty set of labels; \( T_r \) is a transitions relation where \( T_r \subseteq (Q \times A \times Q) \), such that \((q_a, l, q_b) \in T_r\) is a transition with a source \((q_a)\) and sink \((q_b)\) state, and a label \((l)\).

Internal and external actions can be represented in an LTS, but since we are focusing on functional system testing, the transitions will represent interactions between the user and the system (we will refer to \( steps \) as a pair containing one user action and the correspondent system’s response). Annotations are used on the LTS (Annotated LTS, or simply ALTS) to mark these special types of interactions. As a result, the sequences of transitions from ALTS become our test cases.

The simplicity of LTS also allows it to be used as an underlying semantics model for other formalism (e.g. finite state machines), and, consequently, it becomes easier to extend its usage to consider specification of non-deterministic elements and timed models [11]. Furthermore, an LTS is able to visually present the system’s behaviour regarding main and alternative flows of a use case, without requiring much effort in building the model. For an in-depth look and a better understanding of an LTS we refer to Jard and Jéron [12].

3. Specification-based regression testing

Let \( P \) be a baseline version of the program, and \( P' \) be the next version (i.e. delta version) of \( P \). In turn, \( S \) and \( S' \) are, respectively, the baseline and delta specifications for \( P \) and \( P' \). The test suite used to test \( P \) is referred to as \( T \), while \( T' \) is the test suite used to test \( P' \). Throughout this work, \( T \) and \( T' \) will be referred as \( \text{baseline test suite} \) and \( \text{delta test suite} \), respectively.

Our selection strategy (SART) focuses on progressive regression testing, where modifications are performed on a specification model and the goal is to select all test cases that exercise parts of the system that have been modified in \( S' \) (compared to \( S \)). For instance, [3] and [5] state that test cases traversing modifications, named modification-traversing test cases, are more likely to detect regression defects. Two types of modifications are considered here, the addition and removal of model elements (transitions and states) in the specification models. More complex modifications can be expressed as a combination of these two [1, 13]. That being said, those modifications affect the possible paths, and consequently, the scenarios being tested. Accordingly, the test cases for regression testing can be classified as [14]:

- **Obsolete test cases** cannot be executed anymore due to an invalid input/output relationship, or for traversing a removed part of \( S \) or \( P \).
- **Reusuable test cases** exercise unmodified parts of the specification and their correspondent unmodified program construct. Since no modification is exercised, the same result is expected.
- **Retestable test cases** exercise unmodified parts of the specification and may present a different result. For example, test cases exercising unchanged parts of \( S' \) but with new program constructs (e.g. boundary values).
- **New-structural test cases** are structural test cases for new program constructs.
- **New-specification test cases** exercise the modified parts of the specification by executing new code in \( P' \).
Distinguishing these classes of test cases at a system’s specification level can be challenging, because information from source code may not be accessible from specification models (e.g. the program construct). Briand et al. adapted Leung and White’s classification to consider UML designs in order to handle a higher level of abstraction [15]. Similarly, we consider that retestable test cases are sequences of transitions that remain the same but at least one of the labels of those transitions has changed (i.e. no addition or removal of transitions happened, just changes in the label). In turn, unchanged sequences and labels will be considered as reusable test cases, whereas the definition for new-specification remains the same. Here we do not address new-structural test cases because our models contain only behavioural information.

In turn, classification and selection of obsolete test cases is challenging and yet very important. If executed, an obsolete test case will fail not because of a regression defect, but due to an attempt to execute removed parts of the software system. Thus, maintenance to identify and remove these test cases from the test suite is required. Nonetheless, removals can also cause regression defects. For example, an inappropriate removal may cause the SUT to reach a state that should not be reached according to the new specification [1].

By classifying and selecting test cases we alleviate the costs incurred in regression testing, but some scenarios may not be executed and some defects may not be detected. It is important not to compromise defect detection rates, since correcting defects after software release is much more expensive and risky. Therefore, the costs to select and execute the subset have to be smaller than the costs to execute all test cases (i.e. the retest all approach). Otherwise, the retest-all approach is obviously recommended [5].

Unlike other specification-based regression testing techniques, SART applies similarity-based test case selection (STCS) to identify obsolete, reusable and modification-traversing test cases. The goal with STCS is selecting the most diverse test cases based on the assumption that a diverse subset has a higher defect detection rate [6, 7]. This diversity is then obtained by similarity measurements among each pair of test cases. Considering that each test case is a vector of elements (e.g. code statements, model transitions, system conditions, etc.), similarity functions can be used to assign values determining the distance between two vectors. Consequently, close vectors indicate similar test cases. The challenge then becomes choosing appropriate similarity functions and encoding strategies for specific testing contexts [7].

4. A similarity approach for regression testing

The similarity approach for regression testing (SART) is a test case selection technique to automatically identify test cases exercising new, modified, or affected parts of the specification model. In summary, SART compares two sets of test cases from a baseline and a delta version of the specification model. Since test cases are described through steps (i.e. sequences of transitions from the model) comparing the similarities enables testers to identify changes in the model. Our main assumption is that very different sets (i.e. less similar) indicate that severe modifications were performed to a point where the sequences of transitions have significantly changed.

Usage of our selection strategy alone on a pre-defined set of test cases allows automatic selection of the desired subset, but when combined with automatic test case generation, the technique becomes even more powerful since comparison between test cases covering all paths traversing the model can be performed automatically. Before presenting details regarding SART’s execution, we present how the technique can be used in an MBT process (Figure 2).

After changing the functionalities of the system, a new version of the specification is defined, hence a new specification model is obtained. In an MBT context, we assume that there are techniques (either manual or automatic) for creating test cases from the specification model, and since we target high level specification models we provide as input for SART sets of abstract test cases. Usually, these test suites tend to be big and redundant [4], and test case selection is often needed. In order to illustrate how SART selects test cases, we provide an illustrative example.

4.1. Illustrative example

The example is a use case specification for a simple contact list application from a mobile phone (ALTS model in Figure 3). The use case has two scenarios: Add or edit a contact. Editing allows removal of one or several contacts, whilst a new contact can be added by inserting the contact’s information or to import it from a different source (e.g. an e-mail contact, or a social network database).

Eventually, the specification is changed to incorporate three modifications: (1) The deletion of only one contact has been removed. (2) An option to update a contact’s information is added. (3) Export a contact’s information to a different source (e.g. an e-mail). These modifications respectively reflect on the model as follows: (1) Removal of transitions: (4,
Choose one contact and press ‘Remove’ button”,7 and (7, “Selected Contacts are Removed”8); (2) Addition of transitions: (4, “Select one contact and press ‘Update’ option.”9,20) and (21, “Contact’s form is shown”.10,12); (3) Addition of transitions: (16, “Press ‘Save and Export’ buttons”10,22) and (22, “Contact is saved on device and linked accounts”.23).

Based on Korel’s et al. description of interaction patterns from modifications [1, 13], we consider two situations where regression defects can be triggered: (1) the modified element itself can affect software behaviour, or (2) a behaviour specified near a modification can be affected as a side-effect from modifications. Since modifications can affect states, we assume that branching states are sensitive to these modifications because a defect on that branch state can cause the system to reach a different, unexpected state. Consequently, the system will not produce the correspondent output for the performed user action.

In order to address these side-effects, we consider that regions near modifications comprise the modified model elements themselves and the steps from the same level of the modified element. Hence, let $S$ and $S'$ be the baseline and delta version of the LTS model, while $T_{ir}, Q, L$ and $T'_{ir}, Q', L'$ are respectively, the set of transitions, states and labels from $S$ and $S'$. Next we define a modified state ($q_m$), a modified transition ($T_m$) and the region affected by a modification (i.e. set of transitions affected by $q_m$ and $T_m$).

Consider that $q_m \in (Q \cup Q')$, and $T_m$ can either belong to the set of added or removed transitions, named respectively $T_{ir,add}$ and $T_{ir,rem}$.

$$T_{ir,add} = \{T_1, T_2 | T_1 = (q_m, l_a, q_1), T_2 = (q_1, l_b, q_2)\};$$

In other words, the affected region is the set of all pairs of consecutive transitions $T_1, T_2$ such that $T_1$ starts at a modified state ($q_m$). For example in Figure 3, given that state 16 was modified, the affected region will include the transitions (16, “Press the ‘Exit’ button”;19) and (19, “Application is closed”;20) as respectively, $T_1$ and $T_2$. The remainder of affected regions are marked in Figure 3 as solid white edges. Similarly, the modified states ($Q_m$) are shaded and the added and removed transitions ($T_{ir,add}, T_{ir,rem}$) are represented by dotted white edges. In summary, we aim at covering all of them (the shaded background) in our selected subset.

In order to simplify the technique’s step by step execution, we will change the transition’s labels, generating a more compact version of the model (Figure 4 (a)). In addition, we will use the test suites (defined manually by traversing the LTS models) (Figure 4 (b) and (c)).
4.2. SART’s selection strategy

Since each of our abstract test cases are represented as a vector of steps from the ALTS model, a similarity function\(^7\) is used to determine which pair of test cases have similar steps. SART uses an adapted version of the similarity function proposed by Cartaxo et al. [6]. This function was chosen because it presents beneficial results in early executions with SART and with the accompanying selection of test cases generated from ALTS.

Figure 5 presents an overview of SART’s selection strategy that will be next applied in our example. Initially, our selection strategy uses a similarity function to identify the test cases exercising the modifications themselves. Then, we apply test suite minimization techniques to remove unnecessary transition redundancy. Last we add test cases to our subset to enhance transition coverage.

The inputs for SART are \(T\) and \(T'\), and the output is \(T'_s \subseteq T'\), hence no obsolete test cases are selected removing the need for test suite maintenance to identify and remove outdated test cases. The first step is to build the similarity matrix, which contains information between all pairs of test cases \(t_i \, t'_j\) \(\forall t_i \in T, t'_j \in T'\). The baseline test cases are placed in the columns of the matrix, while delta test cases are placed in the rows. Each position \(a[i, j]\) of the matrix is filled with the similarity values calculated through Equation 1.

\[
a[i, j] = \frac{\text{nit}(t_i, t'_j)}{\text{AvgSize}(t_i, t'_j)} / \frac{\text{AvgSize}(t_i, t'_j)}{2}.
\]

The function \(\text{nit}\) counts the number of identical transitions between a test case from \(T\) and \(T'\). Here, identical transitions is a pair of transitions with the same source and sink state, and the same label. This value is divided by an average of sizes (i.e. number of transitions) in order to normalize the ratings among all similarity values.

The resulting value (e.g. 0.80, for \(TC'6\) and \(TC4\)) is then placed in the respective row (6) and column (4) of the matrix. Furthermore, the similarity value “1” indicates that an identical sequence is found in both test suites. Therefore, all transitions are the same and no modification is exercised, being one candidate to be removed from the test suite (that can be seen by

---

\(^7\)To keep the explanation simple, we decided to consider only the set of affected transitions, since affected states can be found through each affected transition.

\(^8\)There are several similarity/distance functions used in literature each with their own strengths and weaknesses, such as the Euclidian, Hamming distance and Jaccard index. Hemmati et al. [7] provide a thorough description and examples for the different similarity functions used for test case selection.
calculating the similarity between $TC1$ and $TC'1$). For the example considered, the similarity values were calculated, resulting in Table 1. The next step is to analyse the similarity values and classify test cases as:

**Obsolete ($T_{obs}$):** Identified through columns that do not have a similarity value of 1 ($TC2$).

**Reusuable ($T_{rea}$):** Rows containing a similarity value of 1 indicate unchanged sequences of transitions already tested in a previous version, thus a reusable test case ($TC7$, $TC8$, $TC10$, $TC11$, $TC12$, $TC14$, $TC15$, $TC16$).

**Targeted ($T_{targ}$):** Contains new specification and also test cases that were not executed before. They can be identified through rows that do not have a similarity value of 1 ($TC2$, $TC3$, $TC4$, $TC5$, $TC6$, $TC9$, $TC13$).

After the classification is concluded, we select test cases that exercise added (targeted test cases) and removed (obsolete) parts of the specification model. Note that an obsolete test case cannot be executed on the SUT, hence SART selects delta test cases very similar to obsolete test cases. This enables execution of similar sequences of paths where a removal has occurred.

First, the delta test cases more similar to each respective obsolete test case are added to the subset. In this example, there is only one obsolete test case ($TC2$), thus, the highest similarity value of the respective column is obtained (0.667), resulting in the selection of $TC1$. Note that $TC1$ exercises a very similar sequence to $TC2$ (both in transitions exercised and in size), especially since $TC1$ also traverses State 4, where a transition’s removal occurred.

Next, we add all targeted test cases to a subset resulting in: $T_{aul} = \{TC1, TC2, TC3, TC4, TC5, TC6, TC9, TC13\}$. As can be seen all modifications have been covered, but note that several test cases repeatedly cover the same transitions (i.e. redundant test cases). The solution is applying test suite minimization to select the minimum set of test cases that cover all transitions of our current subset.

We chose the $H$ heuristic [16] for our minimization step because it has shown good results for revealing defects in an MBT process similar to ours [17]. Firstly, the heuristic defines a cardinality table where each cardinality corresponds to the number of test cases covering a specific test requirement (TR), or in our case, a single transition from the subset. Then, the test cases covering the lowest cardinality TR are included in the reduced subset to ensure coverage of requirements being covered only by a specific test case (named essential test case). As test cases are included, all the respectively covered TRs are marked.

After defining the traceability and cardinality tables (Table 2), we include the test cases covering more requirements from each cardinality set until all requirements are marked. If there is a tie among the test cases, the next cardinality is examined. From our example, we begin with an empty reduced subset $T_r$ and then investigate cardinality 1 for requirements $\{r, f, \}$, containing test cases similar to our reduced subset. By keeping a constant cardinality of each test requirement.

<table>
<thead>
<tr>
<th>TR</th>
<th>Test Cases</th>
<th>Number of Test Cases</th>
</tr>
</thead>
<tbody>
<tr>
<td>$TC1$</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>$TC2$</td>
<td>0.667</td>
<td>0.250</td>
</tr>
<tr>
<td>$TC3$</td>
<td>0.545</td>
<td>0.769</td>
</tr>
<tr>
<td>$TC4$</td>
<td>0.636</td>
<td>0.121</td>
</tr>
<tr>
<td>$TC5$</td>
<td>0.550</td>
<td>0.769</td>
</tr>
<tr>
<td>$TC6$</td>
<td>0.250</td>
<td>0.121</td>
</tr>
<tr>
<td>$TC7$</td>
<td>0.250</td>
<td>0.769</td>
</tr>
<tr>
<td>$TC8$</td>
<td>0.250</td>
<td>0.769</td>
</tr>
<tr>
<td>$TC9$</td>
<td>0.250</td>
<td>0.769</td>
</tr>
<tr>
<td>$TC10$</td>
<td>0.250</td>
<td>0.769</td>
</tr>
<tr>
<td>$TC11$</td>
<td>0.250</td>
<td>0.769</td>
</tr>
<tr>
<td>$TC12$</td>
<td>0.250</td>
<td>0.769</td>
</tr>
<tr>
<td>$TC13$</td>
<td>0.250</td>
<td>0.769</td>
</tr>
<tr>
<td>$TC14$</td>
<td>0.250</td>
<td>0.769</td>
</tr>
<tr>
<td>$TC15$</td>
<td>0.250</td>
<td>0.769</td>
</tr>
<tr>
<td>$TC16$</td>
<td>0.250</td>
<td>0.769</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Test Requirements</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a, b, c, d, e, f, g$</td>
</tr>
<tr>
<td>$h, i, j, k$</td>
</tr>
<tr>
<td>$l, m, n, o$</td>
</tr>
<tr>
<td>$p, q, r$</td>
</tr>
<tr>
<td>$s, t, u, v, w$</td>
</tr>
</tbody>
</table>

In a case study, Bertolino et al. [17] states that the $H$ heuristic reveals more defects when compared to the Greedy (G), Greedy-Essential (GE), and Greedy-1-to-1 Redundancy-Essential (GRE) heuristics.
Table 3: Similarity matrix from the reduced subset and the reusable test cases.

<table>
<thead>
<tr>
<th></th>
<th>TC'7</th>
<th>TC'8</th>
<th>TC'10</th>
<th>TC'11</th>
<th>TC'12</th>
<th>TC'14</th>
<th>TC'15</th>
<th>TC'16</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC'1</td>
<td>0.25</td>
<td>0.25</td>
<td>0.18</td>
<td>0.18</td>
<td>0.18</td>
<td>0.18</td>
<td>0.18</td>
<td>0.18</td>
</tr>
<tr>
<td>TC'4</td>
<td>0.46</td>
<td>0.61</td>
<td>0.43</td>
<td>0.62</td>
<td>0.75</td>
<td>0.66</td>
<td>0.62</td>
<td>0.56</td>
</tr>
<tr>
<td>TC'9</td>
<td>0.5</td>
<td>0.6</td>
<td>0.53</td>
<td>0.76</td>
<td>0.61</td>
<td>0.61</td>
<td>0.76</td>
<td>0.61</td>
</tr>
<tr>
<td>TC'13</td>
<td>0.61</td>
<td>0.61</td>
<td>0.68</td>
<td>0.87</td>
<td>0.87</td>
<td>0.68</td>
<td>0.87</td>
<td>0.87</td>
</tr>
</tbody>
</table>

Figure 6: Goal, Questions and Metrics (GQM) framework defined for our evaluation of SART.

The technique then proceeds by calculating a new similarity matrix (Table 3) between $T_f$ (rows) and $T_{reus}$ (columns). Next, we search for the highest similarity value in each row (random choice is used for tie breaks) and then add the respective column to our final subset followed by the removal of that column from our new matrix in order to avoid repetitive selection of the same set of similarity values. From Table 3 we begin at row $TC'1$ by finding a tie (0.25) between $TC'7$ and $TC'8$, resulting in (random) selection and removal of column $TC'8$. We proceed with analysis of $TC'4$, $TC'9$, $TC'13$ resulting in selection of $TC'12$, $TC'15$, $TC'11$ respectively.

At this point the size limit is reached and SART’s output for our example is: $T_g = \{TC'1, TC'4, TC'9, TC'11, TC'12, TC'13, TC'15\}$. If there were more slots to fill, the technique would return to the first row and repeat the process, until the gaps are filled or all reusable test cases are removed from the matrix. As can be seen, both the modifications and regions shown in Figure 3 are being exercised by our selected subset, increasing our chances of revealing regression defects.

5. Evaluation of our selection strategy

Our goal with this evaluation is to apply SART in an MBT process and measure the trade-off between size reduction and defect detection rate achieved in order to determine whether the technique can be adopted in practice. We execute SART in the same context presented in Figure 2 in order to answer the following research questions: (RQ1) Is it possible to apply SART and still reveal defects with a smaller test suite? (RQ2) How does SART compare with other traditional STCS techniques? To properly organize our evaluation we used the Goal, Questions and Metrics (GQM) framework, presented in Figure 6.

In order to answer those questions we measure coverage and the number of detected defects, allowing us to see whether the technique can reduce the number of test cases, reveal defects and achieve reasonable test coverage. Note that both metrics are measured with respect to the size reduction, i.e. the selected subset. Ideally we need a large sample of specifications to statistically analyse our technique and achieve conclusive results. However, as is common in our field of research, availability of industrial artifacts (e.g. specifications, data about revealed defects) is limited.

Therefore, we divide our evaluation into a case study and an experiment to investigate, respectively, RQ1 and RQ2. First we investigate SART in an MBT selection process where a subject (e.g. a tester) manually selects test cases to test an industrial SUT. Then, we use stochastic generation of models [9] to analyze performance regarding coverage of transitions and modifications in a larger sample of artifacts.

Each evaluation has a different setup and provide complementary information regarding SART’s performance. In summary, the results from our case study allows us to verify whether SART’s automatic selection significantly affects defect detection capability, whereas the experiment provides an overview of SART’s specific coverage capabilities when compared to traditional usage of STCS. All techniques and the experiment were run on the same computer, with 6GB RAM, and an Intel® Core™ i5-2410M processor. The tools, artifacts and methods specific for each evaluation are presented in the respective subsections.

5.1. Case study—Industrial artifacts

Our goal in this case study, is to compare SART’s defect detection capabilities with two different situations in an MBT process: (1) A traditional manual test case selection in an MBT process. (2) Usage of different automatic test case selection techniques. The former allows us to see whether SART has benefits over an expert’s selected subset, whilst the latter allows to compare whether SART has better performance against existing techniques. To simplify our execution and avoid construction validity threats we compare SART only with similarity-based test case selection techniques.

Besides SART, the automatic STCS techniques that we use are different distance measures for similarity-based test case selection, and random selection of test cases (RDM) as a control group. The distance measures are: Counting function (CF), Levenshtein distance (LVS), Jaccard index (JAC), Gower Legendre (GOW) and Sokal Sneath (SOK). We chose those five measures because they are quite disparate and have been used in a previous investigation of STCS [7]. In the end, it led us to compare SART with not only a manual selection but also different automatic test case selection techniques.
Our case study uses artifacts from industry, obtained from a collaboration between practitioners from Ingenico and our research group where an MBT process is used to test a software system that collects and processes biometrics information. We use four specification models (use case templates) that were modified to meet new requirements. As part of our MBT process, we generate ALTS models from those use case templates. Due to confidentiality agreements, we are not able to present the industrial models. Instead, we present the number of states, transitions and modifications in each ALTS to illustrate their size (Table 4). Model 1 and Model 2 are small, whereas Model 3 and Model 4 are bigger and have more complex interactions (e.g. more branches and paths with loops in the ALTS).

We then automatically generate test cases\(^9\) from each ALTS. The generated test suite is then executed in the SUT providing us with the number of detected defects. Finally, we apply all selection techniques to compare two dependent variables: The size of the selected subset, and the number of defects detected when executing the subset.

The graphs in Figure 5.1 presents our results. As can be seen, all automatic selection strategies (SART, CF, LVS, JAC, GOW, SOK and RDM) showed the same results regarding the number of detected defects (Figure 5.1 (a)) and selected a test suite that is either equal or smaller than the one selected by the tester (Figure 5.1 (b)). Moreover, recall was 100% for all selected subset, implying that our case study could not show a significant difference among the investigated techniques.

The tie break between the techniques, in our study, is time to select test cases. For instance, our subject (tester) used four hours to manually select the subsets. Meanwhile, all automatic techniques (SART, CF, LVS, JAC, GOW, SOK and RDM) selected all subsets in less than 200 milliseconds and still revealed the same defects. Thus, the time invested by the tester can be better spent analysing what caused the defects. Besides being a very time consuming process, the manual selection is laborious and tedious, since most test cases have similar sequences that can even be confusing.

In addition, relying on a tester’s expertise can be risky, since human factors such as experience, motivation, etc. can compromise the result leading to an error-prone selection. Usage of an automatic strategy yields more consistent results, whereas the effort lies in deciding whether the technique would be appropriate for the specification being used.

The main contribution of this case study is that SART can be as good as known automatic selection techniques, even though our results are limited to our sample of industrial specification models. Furthermore, they are fairly small ALTS models when compared to large and complex models. Consequently, the small test suites obtained from those models could not fully benefit from using an automatic strategy with coverage criteria, thus hindering comparison between SART and the other STCS techniques. Thus, we decided to execute an experiment using more models and different coverage criteria to identify differences among the selection techniques.

5.2. Experimental study—Investigating coverage

The goal of this experiment is to compare the coverage capabilities of the different selection techniques from our case study. More specifically, we search for evidence indicating whether SART is able to cover modifications performed on a specification model, despite the size reduction. To overcome the limitations in our previous study, we require a large (yet controllable) sample of specification models; but unfortunately, we lack availability of such sample. Therefore, we decided to use a stochastic model generation based on search-based generation of models for technology evaluation [9].

We use the same generator tool described by Oliveira Neto et al. where instances of ALTS models are automatically created and modified [9] based on data from industrial artifacts. That allows us to strike a balance between generalization and statistical power, as recommended by Arcuri and Briand [19] (where the number of artifacts should be at least ten). We create instances of models, named synthetic models, using a generator tool that systematically combines transitions and states in small components named patterns (Figure 8 (a)) that in turn are systematically combined to provide a well-formed ALTS (Figure 8 (b)). After obtaining a baseline layout some transitions are added and removed to obtain a delta version of the specification model (Figure 8 (c)). In order to generate realistic samples of models, our generated sample share characteristics (such as size and layout of states and transitions) extracted from our four industrial models.

\(^9\)We use a simple depth-first search algorithm to traverse all paths of the ALTS under a one-loop-coverage criteria [18].
Then we use automatic test case generation techniques to obtain a large sample of test suites subsequently provided as input to the techniques. Therefore, we enable generation of a numerous and controllable sample because the synthetic models, although artificial, are similar to industrial models. Consequently, we can aim for statistical and practical significance of results by executing the techniques in a larger number of different artifacts, rather than repeatedly executing them on the same set of artifacts.

Our dependent variables comprise two criteria: Transitions and modifications coverage. The former is a widely used criteria to investigate test case selection technique. Moreover, there are known studies focused on transitions coverage for STCS techniques, which helps in comparing SART with the other techniques [20, 7]. In addition, the analysis of modification coverage allows us to see whether our similarity measures are beneficial for identifying modifications in specification-based regression testing.

To complement our analysis of modification coverage, we measure the percentage of selected test cases that exercise the modifications (i.e. targeted vs. reusable test cases), allowing us to observe which techniques are more likely to trigger regression defects. In other words, the hypothesis is that exercising a modification just once may be insufficient to trigger regression defects since interaction of different modified transitions can trigger defects as side effect of a modification [1, 13].

In summary, let $E$ be the set of transitions from the delta ALTS and $E_{cov}$ be the set of transitions covered by the selected subset. Also, $Q_{mod} \subseteq Q$ is the subset of modified states in the ALTS where $q_{mod} \in Q_{mod}$ is a source state of a removed transition or a destination state of an added transition\(^{10}\). Regarding a selected subset $T_{ts}$, let $Q_{mod,ts}, T_{read}, T_{targ}$ be, respectively, the set of modified states exercised by $T_{ts}$, and the sets of reusable and targeted test cases. Thus, we define our three dependent variables as following:

$$V_1 = \frac{|E_{cov}|}{|E|} ; \quad V_2 = \frac{|Q_{mod,ts}|}{|Q_{mod}|} ; \quad V_3 = \left( \frac{|T_{targ}|}{|T_{ts}|} ; \frac{|T_{read}|}{|T_{ts}|} \right)$$

\(^{10}\)By reaching the destination state, we make sure that the added transition is exercised, i.e. covered by the path.
Note that, in our analysis, we will refer to percentages of $V_1$, $V_2$ and $V_3$. In addition, our tools and artifacts comprise the set of industrial specifications (presented in our case study) used to generate the artificial models thorugh the generator tool, while our independent variables are the generated sample (objects), the test case generation algorithm, the generated test suites, and the test case selection technique (factor). Our seven treatments are the techniques used in the case study: SART, CF, JAC, LEV, GOW, SOK and RDM.

Ideally, our dependent variables should also be analysed in terms of rate of defect detection. However, that variable cannot be measured in this experiment because defect data for our synthetic models. One alternative would be to use mutants, however that would provide inaccurate results since synthetic models do not have enough information to elaborate fault hypotheses and place mutants in the model [21].

A prior power analysis reveals that $j = 150$ executions are necessary to achieve statistical significance. In summary, we create $j = 150$ instances of models, each generation yields a pair of ALTS (i.e. baseline and delta) with test suites ($N = 300$ test suites with different quantities of test cases). Ultimately, we execute all 7 techniques 150 times yielding 1050 data points. Results are presented in Figures 9 (a), (b) and (c).

### 5.3. Statistical evaluation

During execution, each synthetic model yields a baseline and delta suite with an average size of, respectively, 14 ($\sigma = 1.95\%$) and 15 ($\sigma = 2.28\%$) test cases, which complies with the set of four specifications that we use as seed for our generator tool.

For all treatments, a test suite with 15 test cases is reduced to an average size of 6 test cases ($\sigma = 1.11\%$), hence reducing the number of test cases by approximately 37%.

At first, $V_1$ (Figure 9 (a)) shows no significant improvement in transition coverage for all STCS techniques when compared to the random selection. Unfortunately, our sample of industrial specification models are either small or medium, hence yielding test suites with a small amount of transitions to be covered. Consequently, by repetitively executing RDM, we can cover most transitions without analysing similarities among test cases. For example, most transitions (an average of 50–65% ) are still being exercised after removing nearly 67% of test cases. However, note that SART has similar results when compared to known STCS techniques with respect to transitions coverage.

Furthermore, results for $V_2$ in Figure 9 (b) confirm the main advantage of SART. Unlike the other remaining STCS and RDM (an average of 65% coverage), SART was the only technique being able to consistently cover all modifications (100%). Both RDM and the similarity-based techniques have varied significantly ($\sigma \approx 20\%$) indicating that they are not reliable when it comes to selecting modification-traversing test cases.

$V_3$ reinforces that evidence by showing (Figure 9 (c)) that most test cases selected by SART exercise the modified and affected parts of the specification since a balanced proportion between Targeted ($\mu = 57.67\%$, $\sigma = 11.92\%$) and Reusable ($\mu = 42.33\%$, $\sigma = 11.92\%$) test cases are selected. The remaining techniques, on the other hand, predominantly select reusable test cases indicating that the modifications or affected parts, even if covered, may not be executed with different combinations of scenarios (or sequences of transitions) during regression testing. Thus, it is less likely to reveal regression defects.

In order to provide more consistent evidence to our visual analysis regarding results from $V_1$ and $V_2$, we use some statistical testing on the collected data. Some of the intervals displayed in Figure 9 (a) and (b) overlap meaning that a statistical test can provide conclusive evidence whether the treatments (techniques) are indeed significantly different regarding our dependent variables. We perform all tests in our data considering a significance level of $\alpha = 0.05$.

In order to obtain statistical evidence regarding the overall difference between all treatments (Table 5), we apply a Friedman test on our data. The result allows us to reject our two null hypotheses, that the treatments have the same performance regarding $V_1$ and $V_2$ (each with $p$-value $< 2.2 E-16$). In addition, we then used pairwise Mann-Whitney test with all pairs of treatments to obtain two vital pieces of information: (i) the statistical difference between each pair of techniques, and (ii) the effect size of that difference, aiming to see whether a specific technique differs significantly from the others.

Therefore, we perform posthoc analysis to obtain effect sizes. We use the Vargha-Delaney’s $A_{12}$ to understand how likely the compaison favours one treatment than the other [23]. So, if we observe the effect size ($A_{12}$) of the first comparison in Table 5, we conclude the effect size in the comparison is 0.926 in favour of SART (CI [0.893, 0.950]), which is a large effect size. Moreover, the $p$-value yields statistically significant difference (SSD) for a 95% confidence level. We obtain similar conclusions when we observe SART in the remaining comparisons (Rows 1 – 6 of Table 5), hence reinforcing the evidence of SART’s capability to cover modifications when comparing to existing similarity-based techniques in literature. In addition, note that the techniques LVS, JAC, GOW and SOK are not significantly different regarding their transition and modification coverage (respectively, $V_1$ and $V_2$).

Based on our visual analysis of Figure 9 (a), one may assume

---

11We used a simple depth-first search (DFS) to traverse all paths of the ALTS (loops included) only once.

12By drawing 15 samples of specifications, the value of $j$ was obtained using $j = \left\lceil\frac{100}{\sigma}\right\rceil$ presented in [22], for $V_1$ and $V_2$.

13Especially for Model 2 and Model 3 in Table 4.

14We choose not to include $V_3$ in our detailed statistical analysis because each technique has two datasets to analyse in $V_3$ regarding coverage of Targeted and Reusable test cases. Consequently, a pairwise comparison of all techniques yields a large number of tests that would increase the complexity of our analysis while not significantly complementing our conclusions from the visual analysis.

15After using normality tests in our dataset we concluded that our data is not normally distributed, thus usage of parametric tests is risky. Moreover, non-parametric tests have been a suitable choice for analysis on different empirical studies on software engineering [19].

16First, we did pairwise comparisons with Bonferroni corrections, but the results presented ties for some comparisons. Thus, to avoid conclusion validity threats due to Bonferroni corrections, we calculate all pairwise comparisons between pairs of treatments.
that the techniques have similar transition coverage capability. However, the posthoc analysis provides us more details regarding $V_1$. Similar to our conclusions regarding $V_2$, no significant difference is seen when comparing LVS, JAC, GOW and SOK. However, we observe a bigger difference between comparisons involving CF. The reason is the bigger variation among measurements of $V_1$ when executing $V_2$. Since the synthetic models were small, the technique had to resolve many tie breaks through random choice. In fact, that reflects on its comparison to RDM by reducing its size to Medium.

The goal with this subsection is to provide detailed information regarding our visual findings under statistical evidence and tests. Ultimately, we mitigate conclusion validity threats when we carefully check our data and avoid relying on assumptions that could lead us to a different direction regarding consistence of our collected data (e.g. usage of parametric tests or ANOVA). Certainly the statistical significance achieved in our analysis is complemented in the next subsection by our discussion regarding practical significance of our findings.

### 5.4. Summary of results

In summary, both evaluations show that SART is able to reduce the number of test cases required to test modified specifications. SART can also reveal defects and spare the tester of doing a visual analysis of the specification to find modifications and then manually select test cases. Even though they are beneficial features of STCS in general, SART specifically ensures coverage of modified parts of the specification by analysing similarities among test cases. In addition, we believe that the automatic classification of test cases into reusable, targeted and obsolete can assist different testing activities, such as maintaining the test suite by removing all obsolete test cases at each version. Therefore, SART’s similarity analysis focused on test cases from different version gives the technique leverage when compared to any other technique simply concerned in selecting a diversity of model elements. Our detailed statistical analysis show robustness and consistency in our usage of similarity-based test case selection to identify modification-traversing test suite.

However, if the specification was not modified, SART is not recommended since our similarity function relies on the assumption that both test suites, provided as input, belong to two different versions of a modified specification. In addition, usage of a matrix can hinder efficiency of our selection, since test suites with thousands of test cases yield over one million similarity values to analyse. To address this issue the implementation can use concurrency to analyse different rows/columns of the matrix simultaneously and then compare the highest values found in each part.

Our results show SART’s advantages already with small models with few modifications. Bigger models that present complex interactions and numerous model elements to be modified may present even better results since the remaining technique do not aim to identify modifications by analysing different model versions. However, an experiment with more complex and bigger models requires careful planning and an experimental design (e.g. full factorial) more complex to execute and analyse.

Although our evaluation focused on a single type of model (ALTS), SART can be used with different types of model. Our similarity function analyses test cases as sequences of labels,
transitions and states; all of them are abstract elements that can be found in many types of models. Even though those elements may be used/named differently in some types of models (e.g., sequence diagrams, activity diagrams, finite state machines), those elements can be interpreted as a vector (test case) and used in our similarity analysis. In addition, some existing techniques are able to create suitable ALTS for SART from other types models, such as sequence diagrams [24].

Similarly, different model elements that interest a tester or belong to a specific type of model (for example, guards in transitions and clocks in real-time system models) can be incorporated in a vector and, in turn, used in the remainder of the similarity-based test case selection process (as described by Hemmati et al. [7]). Then SART could be used to analyse the differences between two versions of the vector to select test cases. For now, our conclusions and applicability of SART are limited to a model-based testing context.

5.5. Threats to validity

The main limitation of our evaluation is that we do not have access to a large sample of industrial specification models and defect data, hence hindering generalization of our results. For our case study, we believe that using more industrial models would show more variation among the number of defects detected by each technique in our case study, enabling statistical analysis regarding defect detection capability. Also, the specifications are not large and complex, hence the set of diverse test cases becomes smaller, whereas larger models would show more redundancy hence being more suitable to use STCS. Nonetheless, the subject participating in our case study emphasized that it was very difficult and tedious to perform manual selection even with the small specifications.

Furthermore, the lack of defect detection analysis on our experiment hinders our conclusions regarding coverage, since a dependent variable analysing detected defects with our synthetic models could complement our coverage analysis by plotting how both coverages relate to, for example, regression defects. Unfortunately, any assumption regarding defects on synthetic models, by itself, creates conclusion and construction validity threats. On the other hand, our results regarding coverage already show statistically significant evidence towards the benefits of using SART to test modified specifications.

For now, we did not compare SART with other regression test case selection techniques because many techniques use different types of models and rely on different assumptions regarding modifications on those models. Thus, comparing those techniques with SART, at this stage, would add several construct and internal validity threats since controlling all those assumptions in an experiment can encumber analysis. First, we wanted strong evidence that SART behaves as known STCS but also targets selection of modification-traversing test cases.

Eventually we intend to do another experiment as more specifications become available, since new models are being created and modified by our industry partner. In addition, we intend to reproduce the experiment described in this paper by increasing the number of synthetic specifications, and perform a full-factorial experiment investigating how the type of model affects SART’s performance.

6. Review on test case selection for regression testing

Test case selection for regression testing is a widely researched topic in literature, resulting in proposal of several techniques, most of them targeting artifacts from source code level [1]. Regarding specification-based approaches, there are several ways to select test cases, each with its own benefits and drawbacks.

A strategy widely used for test case selection in general is a random selection. The subset is chosen by randomly adding test cases until the subset size meets the resource constraints for performing the test [25, 6]. In addition, some experienced testers may use their own expertise or knowledge about the SUT to select a proper set of test cases. In either cases, these strategies are risky for not relying on a formal criterion able to express representativeness of a selected subset, resulting in a proposal of more sophisticated approaches targeting specific coverage criteria.

Another criterion is modification, and identifying modifications by comparing different versions is one of the main strategies for selective regression testing. This comparison, however, can be costly depending on a software’s complexity and size. One of the first techniques with that strategy was proposed by Laski and Szemer [26]. The goal was to compare control flow graphs obtained from different versions of a source code and then identify subgraphs comprising the modified transitions and states, that in turn are mapped to code statements.

Several more recent techniques select regression test cases by identifying model modifications [27, 28]; however, not all modifications are handled by those techniques. For example, some are unable to identify removed elements or more complex modifications (e.g. to replace an architectural module, or change a complex component of the software).

Moreover, some regression defects may be found on unmodified parts of software, triggered as a side effect of a modification, such as software parts dependent on a modified element, leading then to more sophisticated selection strategies where dependency analysis is required. In dependency analysis of specification models [1, 13], all of the model elements are investigated to identify their correspondent dependencies. Discussed initially by Korel et al. [1] for the model-based context, there are three aspects in which modifications can cause defects: the model can be affecting the modification, the modified part can be affected by the model, and a side effect can be introduced by the modification.

Consequently, comparison alone may not be sufficient to ensure safe regression testing in model-based testing, for some of the defects may be hidden under the affected, affecting or side-effect types. Then selection is done by choosing test cases traversing any of those marked dependencies. On the other

---

17 Using a larger sample of more varied industrial models.
hand, those techniques tend to be costly in practice, or limited by constraints (e.g. a system’s size) because complex software systems have several components often dependent among them, and being able to analyse all these components and the possibilities of interactions requires a lot of time and effort. In order to reduce the cost of this analysis, some techniques [29, 30] perform a simpler model analysis, by defining boundaries where the modification can reach other parts of the software system.

Among the selection strategies described above, SART identifies modifications and analyze similarities between test cases to increase coverage of modification-traversing test cases. But unlike other techniques that rely on model comparison, SART applies similarity-based test case selection (STCS) to identify modifications. Currently SART’s input are XML files representing test cases that can be exported from tools, for instance TestLink. Consequently, the technique does not require model files as input. However, we assume that the test cases have behavioural specification of the SUT, such as a basic and alternative flows.

7. Concluding remarks

This paper presented the similarity approach for regression testing (SART) that combines a similarity-based test case selection technique with model-based testing approaches to select test cases exercising modified parts of a specification model. SART is implemented in the LTS-BT tool [31] and the artifacts of our evaluation are available online18 (except for our case study, due to an NDA).

Instead of analysing similarities among test cases belonging to the same test suite, we analyse similarities between test cases of different versions of a software system, to enable selection of modification-traversing test cases. We rely on the assumption that very different pairs of test cases indicate modified sequences of transitions. Based on the similarity values from rows and columns of a matrix, we are able to automatically classify the test cases and then select the ones traversing modified regions of the model.

In our case study, SART detected the same defects as a set of known STCS techniques and our participant’s manually selected subset. Consequently, SART is a feasible and quick alternative for automatic test case selection. Moreover, manual selection of abstract test cases can be daunting and time consuming especially for inexperienced tester, thus providing an automatic selection would allow testers to dedicate more time to analyse test results and find defects.

Even though the other investigated STCS techniques also have those advantages, our experiment provides evidence that SART is a better choice for specification-based regression testing. Overall, no significant difference was found regarding transition coverage capability when comparing all techniques, but SART presented the best (and more consistent) modification coverage (100% of modifications were covered by test cases, opposed to the average 60% of the remaining investigated techniques). Besides, SART constantly exercises the covered modifications by selecting different scenarios (or flows) in which they appear, whereas the other techniques select mostly reusable test cases. On the other hand, if no modification is performed on the specification, SART can be avoided since the test requirements would be different.

That being said, there are still several aspects regarding SART’s applicability that require further empirical investigation. For example, we believe that SART is independent of a specific model type used in an MBT process. By adapting our similarity function to count UML meta-elements (e.g. activities, objects and messages) instead of transitions, the technique can be easily modified to analyse test cases generated from UML models. Furthermore, we intend to extend our evaluation to a larger factorial experiment considering more factors, such as different types of models, modifications and regression test case selection techniques. In addition, we intend to compare SART with different specification-based techniques. The goal then is to complement our findings regarding SART’s unique features as a similarity-based test case selection technique able to identify modification-traversing test cases.

8. Acknowledgments

This work was partly funded by The Knowledge Foundation (KKS) through the project 20130085: Testing of Critical System Characteristics (TOCSYC).

References


18https://sites.google.com/site/fgonetosite/home/downloads

Model-Based Testing of Reactive Systems, Vol. 3472 of Lecture Notes

the automatic synthesis of conformance test cases for non-deterministic reac-
tive systems, International Journal Software Tools for Technology Trans-

tended dependence analysis, in: SOQUA '07: Fourth international work-
shop on Software quality assurance, ACM, New York, NY, USA, 2007,
pp. 62–69.


based on UML designs, Information and Software Technology 51 (1)
(2009) 16–30, special Section - Most Cited Articles in 2002 and Regu-
lar Research Papers.

[16] M. J. Harrold, R. Gupta, M. L. Sofia, A methodology for controlling the
size of a test suite, ACM Transactions Software Engineering Methodology

Corques, Test suite reduction in good order: Comparing heuristics from
a new viewpoint, in: The 22nd IFIP International Conference on Testing

proach, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA,
2006.

[19] A. Arcuri, L. Briand, A hitchhiker’s guide to statistical tests for assessing
randomized algorithms in software engineering, Software Testing, Verifi-

[20] E. G. Cartaxo, Estratégias para controlar o tamanho da suite de teste ger-
ada a partir de abordagens de teste baseado em modelos, Ph.D. thesis,

relevance for software testing, in: Proceedings of the 1994 ACM SIG-
SOFT International Symposium on Software Testing and Analysis, IS-

for Experimental Design, Measurement, Simulation and Modeling, John

[23] A. Vargha, H. D. Delaney, A critique and improvement of the CL
common language effect size statistics of McGraw and Wong, Journal

tion by means of UML sequence diagrams and labeled transition systems,
in: Proceedings of IEEE International Conference on Systems, Man and

empirical study of regression test selection techniques, ACM Transactions

[26] J. Laski, W. Szemer, Identification of program modifications and its ap-
plications in software maintenance, in: ICSM ’92: Proceedings of the
282–290.

approach to select regression test suite to validate changed software, in:
Proceedings of the 2008 15th Asia-Pacific Software Engineering Confer-
ence, APSEC ’08, IEEE Computer Society, Washington, DC, USA, 2008,

regression testing approach for evolving software systems with flexible
tool support, in: Proceedings of the 2010 17th IEEE International Con-
ference and Workshops on the Engineering of Computer-Based Systems,
41–49.

[29] P. K. Chittimali, M. J. Harrold, Regression test selection on system re-
quirements, in: Proceedings of the 1st India Software Engineering Confer-

selection technique, in: Proceedings of the International Conference on
Software Maintenance (ICSM’09), 2009, pp. 515–518.

[31] E. G. Cartaxo, W. L. Andrade, F. G. de Oliveira Neto, P. D. L. Machado,
LTS-BT: A tool to generate and select functional test cases for embed-