Information and Software Technology 80 (2016) 124–137

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

Full modification coverage through automatic similarity-based test case selection

Francisco G. de Oliveira Neto[a,*], Richard Torkar[a,c], Patrícia D.L. Machado[b]

[a] Chalmers and the University of Gothenburg, Sweden
[b] Federal University of Campina Grande, PB, Brazil
[c] Huawei Technologies Sweden AB, Sweden

ARTICLE INFO

Article history:
Received 23 December 2015
Revised 17 July 2016
Accepted 23 August 2016
Available online 24 August 2016

Keywords:
Regression testing
Test case selection
Model-based testing
Experimental study

ABSTRACT

Context: This paper presents the similarity approach for regression testing (SART), where a similarity-based test case selection technique (STCS) 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 the usage of similarity measures for modification-traversing test case selection.

Method: We investigate SART in a case study and an experiment. The case study uses artefacts 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 artefacts 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.

1. Introduction

Regression testing at the code level has been widely investigated in literature [1] and enables solution for most software structure 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.

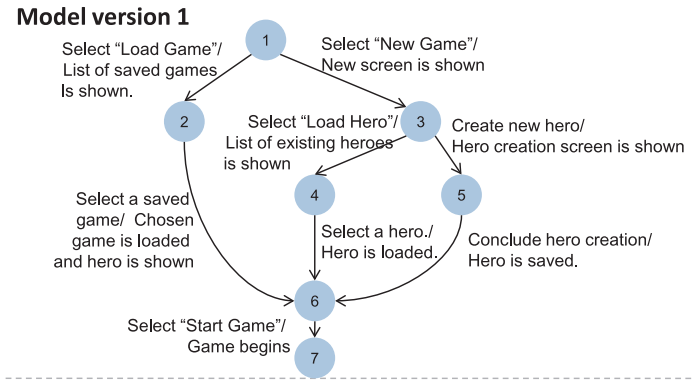
On the other hand, those generated test suites, very often, include numerous redundant test cases that can make it significantly more difficult to execute regression tests [4]. Rather than executing all test cases (i.e. the retest all approach), a more feasible approach to regression testing is to select 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

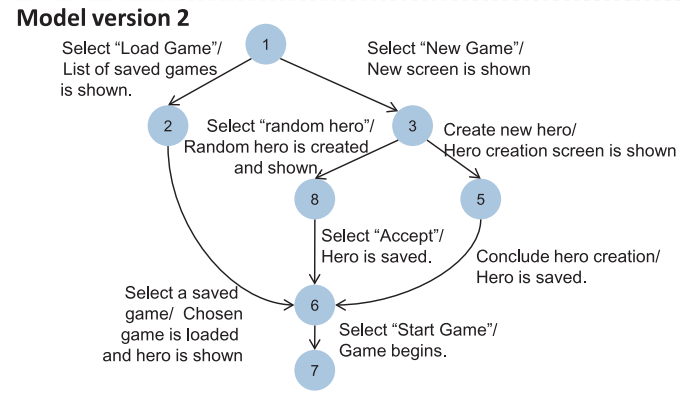
Given the amount of high level information available, the main challenge then becomes selecting a representative subset of test cases in order to reduce the costs of regression testing at the system level. In other words, we aim at maximizing the chances of detecting defects as well as minimizing the number of test cases

* Corresponding author.
E-mail address: francisco.de.oliveira.neto@gu.se (F.G. de Oliveira Neto).

http://dx.doi.org/10.1016/j.infsof.2016.08.008
0950-5849/



TC1	TC2	TC3
Select "Load Game"/ List of saved games is shown	Select "Load Game"/ List of saved games is shown	Select "Load Game"/ List of saved games is shown
Select a saved game/ Chosen game is loaded and Hero is shown in screen.	Select "Load Hero"/ List of existing heroes is shown	Create new hero/ Hero creation screen is shown
Select "Start Game"/ Game begins	Select a hero./ Hero is loaded.	Conclude hero creation/ Hero is saved.
	Select "Start Game"/ Game begins	Select "Start Game"/ Game begins



TC'1	TC'2	TC'3
Select "Load Game"/ List of saved games is shown	Select "Load Game"/ List of saved games is shown	Select "Load Game"/ List of saved games is shown
Select a saved game/ Chosen game is loaded and Hero is shown in screen.	Select "random hero"/ Random hero is created and shown.	Create new hero/ Hero creation screen is shown
Select "Start Game"/ Game begins	Select "Accept"/ Hero is saved.	Conclude hero creation/ Hero is saved.
	Select "Start Game"/ Game begins	Select "Start Game"/ Game begins

Fig. 1. Two versions of specification models and their respective test suites.

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 [3,5,6]. 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 [1,3,7]. As an example, consider the state-based specification models presented in Fig. 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*.[1] If we are unable to 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 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 with 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 the removal of redundancy among test cases [8,9]. 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 [9].

In the scope of model-based regression testing, our contribution is 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. In contrast to other specification-based approaches presented in literature [10], 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.

To better understand the technique we present a case comprising specification, generation and analysis of test cases. However, SART's input can simply be two test suites (e.g. exported from

[1] For each modified state, their connecting transitions were, respectively, removed and added.

[2] State 3 was directly affected by the addition of State 8 because a new transition was added to it.

TestLink[3]), from different versions of the same SUT. Considering our example in Fig. 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.

In order to gather evidence regarding SART's feasibility, we perform a case study and an experiment. First, we analyse SART with industrial artefacts that include specification models, a SUT and the detected defects.[4] Complementary to our case study, we do an experiment where SART selects test cases automatically generated from a larger sample of synthetic models obtained through stochastic generation [11]. We compare SART with other STCS techniques, and both analyses reveal 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 the background to 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 the generation and 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). Although there is a difficulty in tracing abstract test cases to the respective executable code parts, it is easier to identify which functionality or use case scenarios are being tested.

The specification model used in the case study presented in this paper is a labelled 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 Fig. 1) [12]. An LTS is defined as a 4-tuple $S = (Q; A; T_{tr}; q_0)$, 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_{tr} is a transitions relation where $T_{tr} \subseteq (Q \times A \times Q)$, such that $(q_a, l, q_b) \in T_{tr}$ 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). For an in-depth look and a better understanding of an LTS we refer to Jard and Jéron [13].

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 *baseline test suite* and *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, [3,6] 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 [1,14]. That being said, those modifications affect the possible paths, and consequently, the scenarios being tested. Accordingly, Leung and White classify test cases for regression testing as [5]:

Obsolete test cases cannot be executed anymore due to an invalid input/output relationship, or for traversing a removed part of S or P.
Reusable 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. [15] and Fourneret et al. [7] adapted Leung and White's classification to consider UML designs in order to handle a finer grained classification at a higher level of abstraction. Similarly, we adapt the classification to consider that retestable test cases are sequences of steps that remain the same but at least one of the labels of those steps has changed (i.e. no addition or removal of steps 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

[3] TestLink (www.testlink.org) is a tool for test artefact management, and test suites can be exported as XML files.

[4] We here use the word defect as defined by IEEE Std. 29119-1:2013(E)

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 expensive and risky. Therefore, the costs to select and execute the subset have to be smaller than the costs to *retest all*. Otherwise, the retest-all approach is obviously recommended [6].

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 [8,9]. 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, user actions), similarity functions can be used to assign values determining the distance between two vectors, such that close vectors indicate a pair of similar test cases. The challenge then becomes choosing appropriate similarity functions and encoding strategies for specific testing contexts [9].

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 using only information from test cases. 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 (e.g. sequences of transitions from the model) comparing the similarities enables testers to identify changes in the specification (S). Our main assumption is that *very different* sets (i.e. with low similarity) indicate that extensive modifications were performed to a point where the sequences of steps 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 (details in Section 4.3). Before presenting details regarding SART's execution, we present how the technique can be used in an MBT process (Fig. 2).

After changing the functionality 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 Fig. 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 contact list (e.g. e-mail). These modifications respectively impact the model as following: (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."*, 21) and (21, *"Contact's form is shown."*, 12); (3) Addition of transitions: (16, *"Press 'Save and Export' buttons'."*, 22) and (22, *"Contact is saved on device and linked accounts"*., 23).

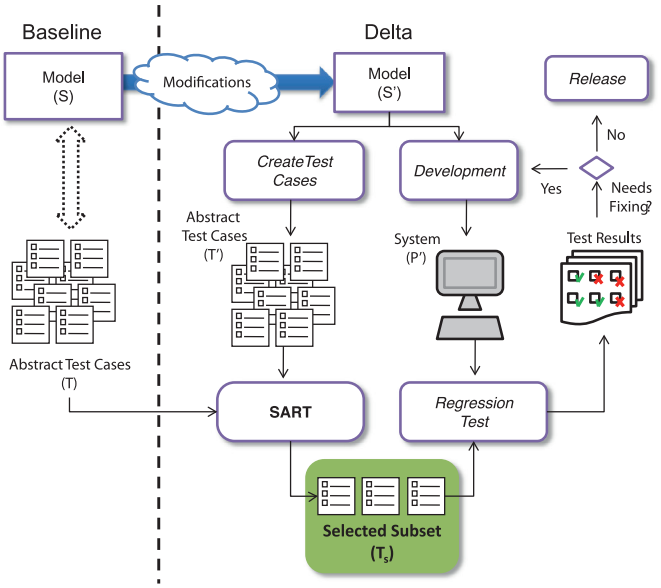


Fig. 2. Example of a test process suitable for SART.

Based on Korel's et al. description of interaction patterns from modifications [1,14], 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 of modifications. Since modifications can affect states, we assume that branching states[5] 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 corresponding 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.[6] Hence, let S and S' be the baseline and delta version of the LTS model, while T_{tr}, Q, L and T'_{tr}, 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 ($\vec{t}_m$) and the region affected by a modification (i.e. set of transitions affected by q_m and $\vec{t}_m$). Consider that $q_m \in (Q \cup Q')$, and $\vec{t}_m$ can either belong to the set of added or removed transitions, named respectively $T_{tr:\,add}$ and $T_{tr:\,rem}$.

$$\vec{t}_m \in T_{tr:add} \iff \vec{t}_m \notin T_{tr} \wedge \vec{t}_m \in T'_{tr};$$
$$\vec{t}_m \in T_{tr:rem} \iff \vec{t}_m \in T_{tr} \wedge \vec{t}_m \notin T'_{tr};$$

Therefore, in order to define the set of affected regions[7] (named $T_{tr:\,reg}$) in S', consider that $\vec{t}_1, \vec{t}_2 \in T'_{tr}$, $q_1, q_2 \in Q'$ and $l_a, l_b \in L'$:

$$T_{tr:reg} = \{\vec{t}_1, \vec{t}_2 \mid \vec{t}_1 = (q_m, l_a, q_1), \vec{t}_2 = (q_1, l_b, q_2)\};$$

[5] States with more than one outgoing transition.

[6] The level is the longest distance between the current and the initial state.

[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.

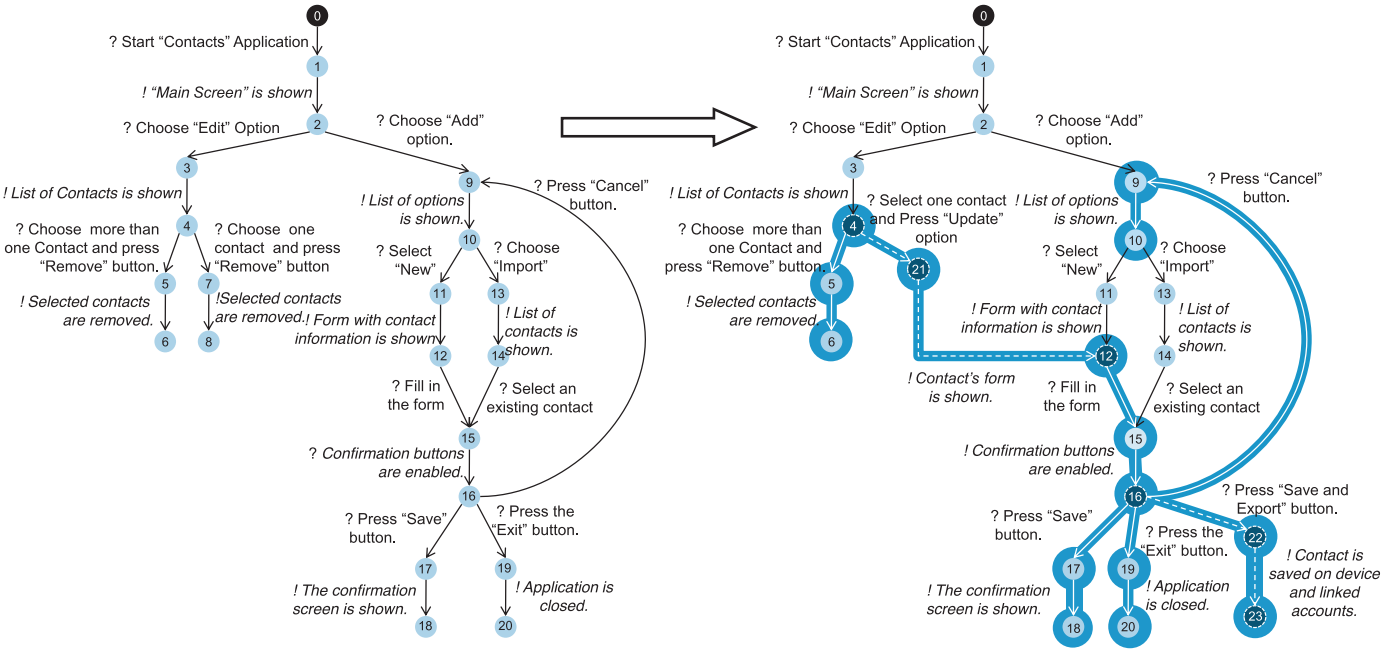


Fig. 3. Examples of ALTS specification models and the model elements affected by model's modifications. Transitions'labels starting with'?'indicate a step, whereas '!' indicate an expected system output.

In other words, the affected region is the set of all pairs of consecutive transitions $\vec{t}_1$, $\vec{t}_2$ such that $\vec{t}_1$ starts at a modified state (q_m). For example in Fig. 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, $\vec{t}_1$ and $\vec{t}_2$. The remaining affected regions are marked in Fig. 3 as solid white edges. Similarly, the modified states (Q_m) are shaded and the added and removed transitions ($T_{tr:\,add}$, $T_{tr:\,rem}$) are represented by dotted white edges. In summary, we aim at covering all of them (the shaded background) in our selected subset.

We changed labels from Fig. 3 to provide a more compact compact version of the model (Fig. 4a). In addition, we will use the test suites (defined manually by traversing the LTS models) (Fig. 4b 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 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. [8] (the original function is presented in Section 5) because it presented beneficial results in early executions with SART and with the accompanying selection of test cases generated from ALTS. Below we present the steps for SART's selection strategy that will be used in our example:

1. Use T and T' to build a similarity matrix;
2. Classify test cases in: reusable, targeted or obsolete;
3. Select test cases: $t'_j \in T'$
   (a) Removals: similar to obsolete test cases;
   (b) Additions: covering new scenarios from S';
4. Remove redundancies through test suite minimisation;
5. Include dissimilar reusable test cases;
6. Export resulting subset T_s;

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 $(t_j, t'_i) \mid t_j \subseteq T, t'_i \subseteq T'$. 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 Eq. (1).

$$a[i,j] = \frac{nit(t'_i, t_j)}{\text{AvgSize}(t'_i, t_j)}; \quad \text{AvgSize}(t'_i, t_j) = \frac{|t'_i| + |t_j|}{2}. \tag{1}$$

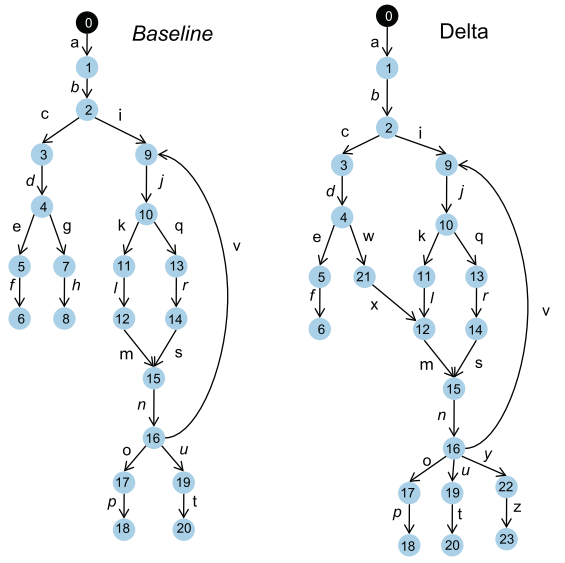
The function *nit* counts the number of identical transitions between a test case from T and T'. Here, identical transitions (or steps) is a pair of transitions with the same source and sink state, and the same label. This value is divided by an average of sizes (i.e. number of transitions) in order 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, so this is a candidate to be removed from the test suite (that can be seen by calculating the similarity between $TC1$ and $TC'1$). Table 1 shows all similarity values calculated in our example.

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 ($TC2$).
Reusable (T_{reus}): *Rows* containing a similarity value of 1 indicate unchanged sequences of transitions already tested in a previous version, thus a reusable test case ($TC'7$, $TC'8$, $TC'10$, $TC'11$, $TC'12$, $TC'14$, $TC'15$, $TC'16$).
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 ($TC'2$, $TC'3$, $TC'4$, $TC'5$, $TC'6$, $TC'9$, $TC'13$).

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



(a) Compact version of the specification model.

Baseline Test Suite - T																
TC1	a	b	c	d	e	f										
TC2	a	b	c	d	g	h										
TC3	a	b	i	j	k	l	m	n	o	p						
TC4	a	b	i	j	q	r	s	n	o	p						
TC5	a	b	i	j	k	l	m	n	v	j	k	l	m	n	u	t
TC6	a	b	i	j	k	l	m	n	v	j	q	r	s	n	u	t
TC7	a	b	i	j	q	r	s	n	v	j	k	l	m	n	o	p
TC8	a	b	i	j	q	r	s	n	v	j	q	r	s	n	o	p

(b) Baseline test suite.

Delta Test Suite – T'																
TC'1	a	b	c	d	e	f										
TC'2	a	b	c	d	w	x	m	n	y	z						
TC'3	a	b	c	d	w	x	m	n	v	j	k	l	m	n	y	z
TC'4	a	b	c	d	w	x	m	n	v	j	q	r	s	n	o	p
TC'5	a	b	i	j	k	l	m	n	y	z						
TC'6	a	b	i	j	q	r	s	n	y	z						
TC'7	a	b	i	j	k	l	m	n	o	p						
TC'8	a	b	i	j	q	r	s	n	o	p						
TC'9	a	b	i	j	q	r	s	n	u	t						
TC'10	a	b	i	j	k	l	m	n	v	j	k	l	m	n	u	t
TC'11	a	b	i	j	k	l	m	n	v	j	q	r	s	n	u	t
TC'12	a	b	i	j	k	l	m	n	v	j	q	r	s	n	o	p
TC'13	a	b	i	j	k	l	m	n	v	j	q	r	s	n	y	z
TC'14	a	b	i	j	q	r	s	n	v	j	q	r	s	n	o	p
TC'15	a	b	i	j	q	r	s	n	v	j	k	l	m	n	u	t
TC'16	a	b	i	j	q	r	s	n	v	j	k	l	m	n	o	p

(c) Deta test suite.

Fig. 4. Simplified artefacts obtained from our example.

be executed on the SUT, hence SART chooses to select 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 most similar to each respective obsolete test case are added to the subset. In this example, there is only one obsolete test case (*TC*2), thus, the highest similarity value of the respective column is obtained (0.667), resulting in the selection of *TC*′1. Note that *TC*′1 exercises a very similar sequence to *TC*2 (both in transitions exercised and in size), especially since *TC*′1 also traverses State 4, where a transition's removal occurred.

Table 1
Similarity matrix from test suites of Fig. 4. Note that the first row presents an original unmodified test case.

	*TC*1	*TC*2	*TC*3	*TC*4	*TC*5	*TC*6	*TC*7	*TC*8
TC′1	1	0.667	0.250	0.250	0.182	0.182	0.182	0.182
TC′2	0.500	0.500	0.400	0.300	0.308	0.308	0.231	0.231
TC′3	0.545	0.545	0.769	0.231	0.625	0.625	0.563	0.375
TC′4	0.364	0.364	0.538	0.692	0.438	0.625	0.688	0.688
TC′5	0.500	0.500	0.500	0.700	0.769	0.615	0.769	0.385
TC′6	0.250	0.250	0.500	0.800	0.385	0.615	0.615	0.615
TC′7	0.250	0.250	1	0.700	0.615	0.615	0.769	0.538
TC′8	0.250	0.250	0.700	1	0.385	0.615	0.769	0.769
TC′9	0.250	0.250	0.500	0.800	0.538	0.769	0.615	0.615
TC′10	0.182	0.182	0.615	0.385	1	0.813	0.688	0.500
TC′11	0.182	0.182	0.615	0.385	0.813	1	0.875	0.688
TC′12	0.182	0.182	0.769	0.769	0.688	0.875	1	0.813
TC′13	0.182	0.182	0.615	0.615	0.688	0.875	0.875	0.688
TC′14	0.182	0.182	0.538	0.769	0.500	0.688	0.813	1
TC′15	0.182	0.182	0.615	0.615	0.750	1	0.875	0.688
TC′16	0.182	0.182	0.769	0.769	0.688	0.875	1	0.813

Table 2
Traceability between test requirements and test cases.

TR	Test cases								Number of test cases
	TC'1	TC'2	TC'3	TC'4	TC'5	TC'6	TC'9	TC'13	
a	x	x	x	x	x	x	x	x	8
b	x	x	x	x	x	x	x	x	8
c	x	x	x	x	–	–	–	–	4
d	x	x	x	x	–	–	–	–	4
e	x	–	–	–	–	–	–	–	1
f	x	–	–	–	–	–	–	–	1
i	–	–	–	–	x	x	x	x	4
j	–	–	x	x	x	x	x	x	6
k	–	–	x	–	x	–	–	x	3
l	–	–	x	–	x	–	–	x	3
m	–	x	x	x	x	–	–	x	5
n	–	x	x	x	x	x	x	x	7
o	–	–	–	x	–	–	–	–	1
p	–	–	–	x	–	–	–	–	1
q	–	–	–	x	–	x	x	x	4
r	–	–	–	x	–	x	x	x	4
s	–	–	–	x	–	x	x	x	4
t	–	–	–	–	–	–	x	–	1
u	–	–	–	–	–	–	x	–	1
v	–	–	x	x	–	–	–	x	3
w	–	x	x	x	–	–	–	–	3
x	–	x	x	x	–	–	–	–	3
y	–	x	x	–	x	x	–	x	5
z	–	x	x	–	x	x	–	x	5

Next, we add all targeted test cases to a subset resulting in: $T_{aux} = \{TC'1,\ TC'2,\ TC'3,\ TC'4,\ TC'5,\ TC'6,\ TC'9,\ TC'13\}$. 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 [3].

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 (Tables 2 and 3), we include the test cases covering most requirements from each cardinality set until all requirements are marked. If

Table 3
Cardinality of each test requirement from Table 2.

Cardinality	1	3	4	5	6	7	8
TR	e, f, o, p, t, u	k, l, v, w, x	c, d, i, q, r, s	m, y, z	j	n	a, b

Table 4
Similarity matrix from the reduced subset and the reusable test cases.

	$TC'7$	$TC'8$	$TC'10$	$TC'11$	$TC'12$	$TC'14$	$TC'15$	$TC'16$
$TC'1$	0.25	0.25	0.18	0.18	0.18	0.18	0.18	0.18
$TC'4$	0.46	0.61	0.43	0.62	0.75	0.68	0.62	0.56
$TC'9$	0.5	0.8	0.53	0.76	0.61	0.61	0.76	0.61
$TC'13$	0.61	0.61	0.68	0.87	0.87	0.68	0.87	0.87

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 $\vec{e}, \vec{f}, \vec{o}, \vec{p}, \vec{t}, \vec{u}$. The test cases covering the TR at this cardinality are $TC'1$, $TC'4$, $TC'9$, resulting in an addition of $TC'4$ for covering more TRs among them (last row of Table 2). Consequently, TRs $\vec{a}, \vec{b}, \vec{c}, \vec{d}, \vec{w}, \vec{x}, \vec{m}, \vec{n}, \vec{v}, \vec{j}, \vec{q}, \vec{r}, \vec{s}, \vec{n}, \vec{o}, \vec{p}$ are all marked as covered — see Column $TC'4$ in Table 2. Continuing with unmarked TRs at cardinality 1, test cases $TC'9$ and $TC'1$ are added to the reduced subset. The next cardinality is 3 with unmarked TRs $\vec{k}, \vec{l}$ resulting in choice of $TC'13$. After this, all TRs become marked concluding our minimization stage with subset: $T_r = \{TC'4, TC'1, TC'9, TC'13\}$.

So far, all transitions of the subset have been covered with half the number of test cases. However, as mentioned earlier, some regression defects may be triggered as side-effects from nearby modifications. In order to cover the side-effect regions, we fill the gaps left from removing redundant test cases with reusable test cases similar to our reduced subset. By keeping a constant similarity analysis, we ensure that our test cases are still *near* the modifications, even if not covering the modifications themselves.

The technique then proceeds by calculating a new similarity matrix (Table 4) between T_r (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 4 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: $T_s = \{TC'1,\ TC'4,\ TC'9,\ TC'11,\ TC'12,\ TC'13,\ TC'15\}$. 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 Fig. 3 are being exercised by our selected subset, increasing our chances of revealing regression defects.

4.3. Combining automatic test case generation and selection

Test case selection refers to selection of a subset of test cases according to specific criteria. Therefore, selection assumes the existence of an initial set of test cases. There are different strategies to obtain an initial test suite that vary from complete manual creation of test cases to using automatic tools to create test cases from software artefacts. In MBT, automatic test case generation is widely used since information is provided on model artefacts. In turn, model-based regression testing benefits from automatic regeneration of test suites from modified models so that model modifications are reflected in the generated test suite [1,14].

Therefore, automatic test case generation can improve SART's performance if the test suites are generated from different versions of the same model (S and S'). Since test case generation often relies on a coverage criteria to traverse models [18], using different coverage criteria when generating regression test cases may yield inconsistencies in test artefacts, thus risking to also affect the selection technique's performance. For instance, if a tester uses different criteria to obtain T and T', SART will classify some test cases in T' as targeted even if they do not cover modifications, simply because they were not executed in S due to usage of a different criteria to generate T.

SART is implemented in an updated version of LTS-BT [19] along with other test case generation, prioritisation, minimisation techniques from different researchers. Therefore, it can easily be combined with automatic test case generation based on full transitions coverage. Nonetheless, we reinforce that SART receives as input test suites T and T' assuming that they represent test suites from different versions of the SUT.

Due to our proposed MBT context (Fig. 2), we present SART's selection, evaluation and conclusions correlating them to model elements (states and transitions), but similarity functions are general [9]. Thus, Eq. 1 can be used with different test information (e.g. post-conditions, pre-conditions, user actions). Note that ideally, any test case selection technique must execute independent of the generation strategy assuming only the existence of a test suite [3,18]. Nonetheless, literature encourages to combine test case generation and selection techniques especially if the generated test suite is highly redundant, which is also ideal for similarity-based test case selection [8,18].

5. Evaluation of our selection strategy

Our goal with this evaluation is to 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 define our evaluation using the Goal, Questions and Metrics (GQM) framework in order to answer RQ1 and RQ2.

- **Goal:** Evaluate and compare SART's performance
- **Questions:**
  - **RQ1:** Can we use SART and still reveal defects with a smaller test suite?
  - **RQ2:** How does SART compare with other STCS techniques?
- **Metrics:**
  - Defect detection capability;
  - Size reduction;
  - Coverage criteria;

By measuring coverage and the number of detected defects, we can see whether the technique reduces the number of test cases, reveals defects and achieves reasonable test coverage. Note that both metrics are measured with respect to the size reduction, i.e. the selected subset. We also discuss *safety, precision, generality* and *efficiency* of techniques, as suggested in [6,7,9]. Ideally we need a large sample of specifications to statistically analyse the investigated techniques and achieve conclusive results. However, as is common in our field of research, availability of industrial artefacts (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 [11] to analyse performance regarding coverage of transitions and modifications in a larger sample of artefacts.

Each evaluation has a different setup and provides complementary information regarding SART's performance. All techniques and the experiment were run on the same computer, with 6*GB* RAM, and an *Intel® Core™ i5-2410M* processor. Application of all the techniques investigated in this section is simple and straightforward since they are based on well defined mathematical formulas and algorithms existing and provided in literature. Therefore, there are no complex configuration of an experimental environment, technique's dependencies or training to the experimenters.

5.1. Case study–Industrial artefacts

We want to compare SART's defect detection capabilities with two different situations in an MBT process: (*i*) A traditional manual test case selection in an MBT process; (*ii*) 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 than existing techniques. To simplify our execution and avoid construction validity threats we compare SART only with similarity-based test case selection techniques.

5.1.1. Similarity measures

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 [9]. In the end, it led us to compare SART with not only a manual selection but also different automatic test case selection techniques.

SART is an adaptation of the counting function (CF) proposed by Cartaxo et al. [8] that counts the number of repeated steps in a test case divided by the average size of the test cases (see Eq. (2)). Unlike the other measures used in this study, CF and SART consider both the content and length of a test case. The main difference is that SART counts repeated steps on test cases of different versions of a test suite. Considering $t'_i, t'_j \in T'$:

$$sim(t'_i, t'_j) = \frac{nit(t'_i, t'_j)}{\text{AvgSize}(t'_i, t'_j)};\ \text{AvgSize}(t'_i, t'_j) = \frac{|t'_i| + |t'_j|}{2}. \quad (2)$$

We choose the Levenshtein distance (LVS) to compare SART to a technique targeting textual information on test cases. LVS is based on the edit distance between two strings[8]. We use strings from test cases to calculate the edit distance, the resulting value is then the similarity degree. Comparison between LVS and the remaining techniques is important since our abstract test cases are written in natural language, and other techniques (SART included) consider different aspects besides the text information, such as the set of steps removing all redundancies, or unchanged steps from previous versions of the SUT.

Additionally, we compare SART to techniques that analyse information besides textual description of test cases. JAC, GOW and SOK belong to a family of measures based on differences and commonalities between information. In our case, the input is a pair of test cases and their corresponding transitions (either a user action or a system's expected result). As presented in [9], the formula to calculate similarity using JAC, GOW and SOK is:

$$sim(t'_i, t'_j) = \frac{|t'_i \cap t'_j|}{|t'_i \cap t'_j| + w(|t'_i \cup t'_j| - |t'_i \cap t'_j|)} \quad (3)$$

[8] We refer to operations in string to transform t'_1 in t'_2 by changing (c), removing (r) or adding (a) a char. The distance is the sum of operations performed.

Table 5
Data regarding industrial artefacts informing the number of states (S), transitions (T), additions (Add) and removals (Rem), and the number of delta test cases ($|T'|$) created for each model.

Models	Baseline version		Delta version		Modifications			$\|T'\|$
	S	T	S	T	Rem	Add	Total	
Model 1	11	11	10	9	3	1	4	3
Model 2	22	24	20	21	5	2	7	14
Model 3	32	33	28	28	6	1	7	16
Model 4	32	38	22	25	13	0	13	67

The three measures differ on the value assigned to w (Eq. 3) indicating the different weights given to the amount of information differing between test cases[9]. The indices values for JAC, GOW and SOK are, respectively, $w = 1,\ w = 1/2,\ w = 2$. Consequently, each function yields a different similarity value such that, for the same pair of test cases provided, $GOW > JAC > SOK$. We choose those three indices to enable analysis of the nuances of the differences between test cases (in complement to the similarity analysis incorporated in SART, simply comparing past information with the new one).

Unlike SART, none of the measures above compare test cases from T' to test cases from T. As an example, consider $t'_1 = [a, b, c, d, b, c, e]$ and $t'_2 = [a, b, x, y, z]$, such that $|t'_1 \cup t'_2| = 8$, $|t'_1 \cap t'_2| = 2$. Then[10]:

$$\begin{aligned} SART(t'_1, t'_2) &= 2/[(7+5)/2] \approx 0.33 \\ CF(t'_1, t'_2) &= 2/[(7+5)/2] \approx 0.33 \\ LVS(t'_1, t'_2) &= 3c + 2a = 5 \\ JAC(t'_1, t'_2) &= 2/[2 + (1 \times 6)] \approx 0.25 \\ GOW(t'_1, t'_2) &= 2/[2 + (0.5 \times 6)] \approx 0.40 \\ SOK(t'_1, t'_2) &= 2/[2 + (2 \times 6)] \approx 0.14 \end{aligned}$$

5.1.2. Objects and execution

Our case study uses artefacts 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 during release of a new version of the software system 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 5). 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).

From each ALTS, we automatically generate test cases,[11] in turn, manually executed by practitioners. All execution data (reports of failures and defects) was provided to our case study. Therefore, we do not alter or instrument any of the objects, rather they are just input to the techniques. Finally, we apply all selection techniques on the test cases and compare two dependent variables: The size of the selected subset, and the number of defects detected.

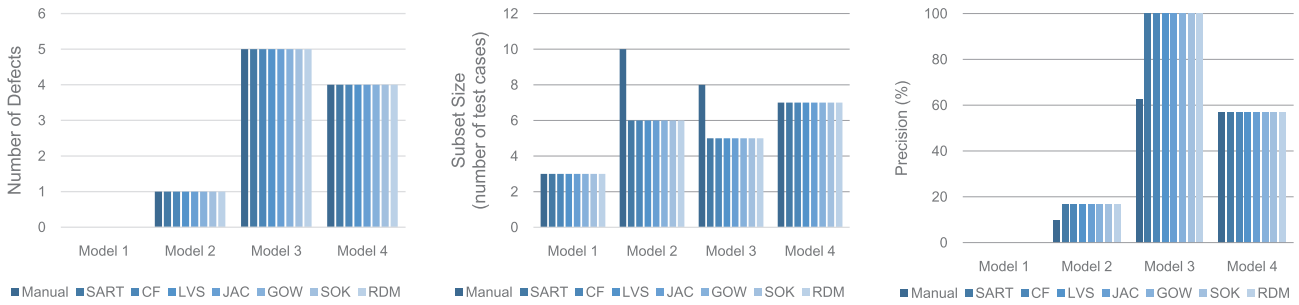
5.1.3. Results and analysis

The graphs in Fig. 5 presents our results, discussed under the perspective of safety, precision, generality and efficiency [6]. As

[9] Note that $|t'_i \cup t'_j| - |t'_i \cap t'_j|$ is the number of steps differing between t'_i and t'_j, i.e. all information of both test cases minus the commonalities.

[10] Note that SART is only different from CF if it analyses similarities between $t_i \in T$ and $t'_j \in T$.

[11] We use a simple depth-first search (DFS) algorithm to traverse all paths of the ALTS under a one-loop-coverage criteria [18].



(a) Number of detected defects. (b) Sizes of selected subset. (c) Precision (defects) of selected subsets.

Fig. 5. Comparison between investigated selection techniques using industrial artefacts.

can be seen, all automatic selection strategies (SART, CF, LVS, JAC, GOW, SOK and RDM) have similar results, because they: (*i*) reveal the same number of detected defects (Fig. 5a), (*ii*) select a test suite that is equal or smaller than the manually selected test suite (Fig. 5b), and (*iii*) are only more precise than manual selection for models 2 and 3 (Fig. 5c).

Moreover, recall for modifications is 100% for all selected subset except for Model 1, but there is no significant difference among the techniques. In addition, T' is relatively small in all of them and several modifications were made, meaning that the majority of the test cases cover modifications. Therefore, in practice, a retest all approach could be more beneficial. In turn, the *precision* of techniques vary according to the model. For instance, a high precision is seen in Model 3 (100% precision) opposed to a low precision in Model 2.

Results for safety and precision vary based on models because their corresponding test suites are still too few and small to provide general results for safety and precision. Therefore the *generality* of our case study is small since our availability of artefacts is too limited (in quantity and diversity[12]) to allow generalization.

Since all techniques present a similar performance, we measure *efficiency* based on the 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 in test report and analysis. 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.

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 and lack generalisation. Moreover, our objects 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 decide 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 decide to use a stochastic model generation based on search-based generation of models for technology evaluation [11].

We use the same generator tool described by Oliveira Neto et al. where instances of ALTS models are automatically created and modified [11] based on data from industrial artefacts. That allows us to strike a balance between generalization and statistical power, as recommended by Arcuri and Briand [20] (where the number of artefacts 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. 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. For a detailed description refer to [11].

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 [11]. Consequently, we can aim for statistical and practical significance of results by executing the techniques in a larger number of different artefacts, rather than repeatedly executing them on the same set of artefacts.

Our dependent variables are: 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 [8,9]. In addition, the analysis of modification coverage allows us to see whether the 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 [1,14].

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.[13] Regarding a selected subset T_{ts},

[12] Ideally, we would like to experiment on numerous test suites from different domains.

[13] By reaching the destination state, we make sure that the added transition is exercised, i.e. covered by the path.

let $Q_{mod:\,ts}$, T_{reus}, 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:

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

Note that, in our analysis, we will refer to percentages of V_1, V_2 and V_3. In addition, our tools and artefacts comprise the set of industrial specifications (presented in our case study) used to generate the artificial models through the generator tool, while our independent variables are the generated sample (objects), the test case generation algorithm,[14] 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, LVS, 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 is unknown 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 [20]. Additionally, our dependent variables allow analysis only under *safety* and *generality*. Since all techniques take milliseconds to execute, analysis under *efficiency* has no practical significance, as observed in the case study. We do not analyse *precision* since we argue in Section 4 that selection of reusable test cases can be beneficial at our level of abstraction. Instead, we focus on statistical and practical significance of our findings.

A prior power analysis[15] 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 Fig. 6a–c.

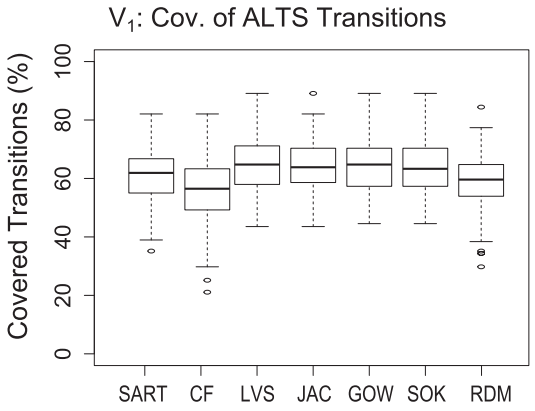
5.3. Statistical evaluation

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.[16] 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%.

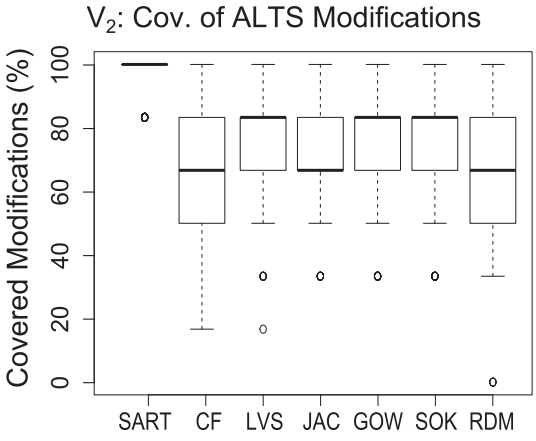
At first, V_1 (Fig. 6a) shows no significant improvement in transition coverage for all STCS techniques when compared to the random selection. Unfortunately, our sample of industrial specification models are either small or medium, hence yielding test suites with a small number of transitions to be covered. 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 Fig. 6b 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%, thus being a *safe technique*). 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.

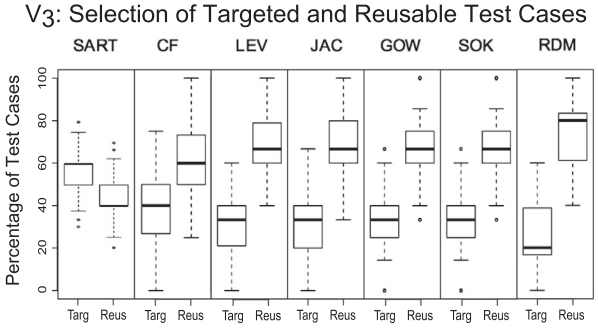
V_3 reinforces that evidence by showing (Fig. 6c) that most test cases selected by SART exercise the modified and affected parts of the specification since a balanced proportion between Targeted ($\mu = 57.67\%, \sigma = 11.92\%$) and Reusable ($\mu = 42.33\%, \sigma = 11.92\%$) test cases are selected. The remaining techniques, on the other hand, predominately select reusable test cases indicating that the modifications or affected parts, even if covered, may not be executed with different combinations of scenarios during regression testing. Thus, it is less likely to reveal regression defects.



(a) Transition coverage (V_1).



(b) Modification coverage (V_2).



(c) Proportion between targeted and reusable (V_3).

Fig. 6. Boxplots for the investigated dependent variables.

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

[15] By drawing 15 samples of specifications, the value of j was obtained using $j = \lfloor \left(\frac{100\sigma}{r\mu_j}\right)^2 \rfloor$ presented in [21], for V_1 and V_2.

[16] Especially for Model 2 and Model 3 in Table 5.

Table 6
Summary of the posthoc statistical analysis for all possible comparisons between treatments. Each analysis determines whether the treatments have a statistically significant difference (*SSD) according to their corresponding *p*-values and effect sizes (S - small; M - medium; L - large).

		Transition coverage - V1						Modification coverage - V2					
Pairwise comparison		*p*-value	$\hat{A}_{12}$	Effect size (CI)	Best	Effect size	SSD∗	*p*-value	$\hat{A}_{12}$	Effect size (CI)	Best	Effect size	SSD∗
1	SART x CF	1.60E-06	0.658	[0.594, 0.718]	SART	M	Yes	2.20E-16	0.926	[0.893, 0.950]	SART	L	Yes
2	SART x RDM	0.02358	0.575	[0.510, 0.638]	SART	S	Yes	2.20E-16	0.964	[0.938, 0.979]	SART	L	Yes
3	SART x LVS	0.00946	0.414	[0.351, 0.479]	LVS	S	Yes	2.20E-16	0.909	[0.873, 0.936]	SART	L	Yes
4	SART x JAC	0.004457	0.405	[0.343, 0.471]	JAC	S	Yes	2.20E-16	0.892	[0.852, 0.921]	SART	L	Yes
5	SART x GOW	0.003329	0.402	[0.340, 0.468]	GOW	S	Yes	2.20E-16	0.896	[0.858, 0.925]	SART	L	Yes
6	SART x SOK	0.003085	0.402	[0.339, 0.467]	SOK	S	Yes	2.20E-16	0.895	[0.857, 0.925]	SART	L	Yes
7	CF x RDM	0.002445	0.399	[0.337, 0.465]	RDM	M	Yes	7.59E-04	0.586	[0.522, 0.648]	CF	S	Yes
8	CF x LVS	4.83E-11	0.285	[0.230, 0.347]	LVS	L	Yes	0.1046	0.447	[0.386, 0.511]	LVS	S	No
9	CF x JAC	2.00E-11	0.281	[0.227, 0.343]	JAC	L	Yes	0.01206	0.419	[0.358, 0.482]	JAC	S	Yes
10	CF x GOW	1.41E-11	0.279	[0.225, 0.341]	GOW	L	Yes	0.00603	0.412	[0.351, 0.475]	GOW	S	Yes
11	CF x SOK	9.28E-12	0.278	[0.223, 0.339]	SOK	L	Yes	0.0272	0.429	[0.367, 0.492]	SOK	S	Yes
12	RDM x LVS	6.05E-06	0.35	[0.291, 0.415]	LVS	M	Yes	5.02E-07	0.354	[0.296, 0.416]	LVS	M	Yes
13	RDM x JAC	2.15E-06	0.343	[0.284, 0.408]	JAC	M	Yes	1.70E-09	0.32	[0.265, 0.381]	JAC	L	Yes
14	RDM x GOW	1.30E-06	0.34	[0.281, 0.404]	GOW	M	Yes	4.35E-10	0.313	[0.258, 0.374]	GOW	L	Yes
15	RDM x SOK	1.08E-06	0.339	[0.280, 0.403]	SOK	M	Yes	1.62E-08	0.333	[0.276, 0.394]	SOK	M	Yes
16	LVS x JAC	0.8442	0.493	[0.428, 0.559]	JAC	S	No	0.3301	0.469	[0.406, 0.532]	JAC	S	No
17	LVS x GOW	0.7208	0.488	[0.423, 0.553]	GOW	S	No	0.2362	0.462	[0.400, 0.525]	GOW	S	No
18	LVS x SOK	0.7839	0.491	[0.426, 0.556]	SOK	S	No	0.5173	0.479	[0.417, 0.542]	SOK	S	No
19	JAC x GOW	0.874	0.495	[0.430, 0.560]	GOW	S	No	0.8113	0.492	[0.430, 0.555]	GOW	S	No
20	JAC x SOK	0.9467	0.498	[0.433, 0.563]	SOK	S	No	0.7501	0.51	[0.447, 0.573]	JAC	S	No
21	GOW x SOK	0.9345	0.503	[0.438, 0.568]	GOW	S	No	0.5873	0.517	[0.455, 0.580]	GOW	S	No

In order to provide more consistent evidence to our visual analysis regarding results from V_1 and V_2,[17] we use some statistical testing on the collected data. Some of the intervals displayed in Fig. 6a 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 6), 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 $< 2.2E-16$). In addition, we then used pairwise Mann-Whitney test with all pairs of treatments to see: (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 $\hat{A}_{12}$ to understand how likely the comparison favours one treatment than the other [22]. So, if we observe the effect size ($\hat{A}_{12}$) of the first comparison in Table 6, 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 6), hence reinforcing the evidence of SART's capability to cover modifications compared 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 Fig. 6a, one may assume 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. 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 effect to Medium.

The goal of this subsection is to provide detailed information regarding our visual findings under statistical evidence and tests. Therefore, our conclusions for the experiment comply with *generality* when considering the limitations from construction validity threats discussed at the end of this section. Ultimately, we mitigate conclusion validity threats by carefully checking our data and avoid relying on assumptions that could lead us to a different conclusion 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. Interpretations of results

In summary, both evaluations provide answers to our research questions. Regarding RQ1, the case study shows that *SART's selected subset still reveals defects even with a smaller test suite*. However, note that, in practice, reduction is not significant since our industrial test suites do not have numerous test cases. In turn, the experiment provide answers to RQ2 such that *SART has similar transition coverage than the other STCS techniques, but it excels in selecting modification-traversing test cases*. Moreover, we achieve statistical significance of our evidence through a rigorous statistical analysis of data. Nonetheless, our experiment is subject to threats to validity, further discussed in our next subsection. But first, we provide interpretation of our results by exposing SART's pros and cons found during our evaluation.

SART's similarity analysis focused on test cases from different versions provides a leverage when compared to other techniques simply concerned with selecting a diversity of model elements. Our detailed statistical analysis show robustness and consistence 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.

[17] We choose not to include V_3 in our statistical analysis to keep a feasible number of pairwise comparisons in our analysis.

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 techniques 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 vectors (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 [23].

Nonetheless, our results yield more research questions. For instance, is there a specific type of modification (addition or removals) that is more sensitive to SART's selection strategy? Is there a threshold regarding the subset size (or number of modifications covered) where SART starts to loose performance when compared to other techniques? For instance, Cartaxo et al. observed that CF is only beneficial for reductions up to 80% of the original test suite size [8]. Now that we have evidence regarding SART's safety, we can compare it with other safe techniques to measure the dependences and costs in applying different techniques that select modification-traversing test cases.

5.5. Threats to validity

Even though we achieve statistically significant difference in our experiment analysis, our evaluation methodology has many threats to validity. Most of them are related to our constructs given the limited availability (quantity and variety) of industrial artefacts. Eventually we intend to do another experiment as more specification models become available. In addition, we intend to reproduce the experiment described in this paper by increasing the number of synthetic specifications,[18] and perform a full-factorial experiment investigating how does the type of model affects SART's performance.

5.5.1. Construction validity threats

Our main construction validity threat is the lack of defect detection analysis on our experiment. That 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. As a countermeasure we enhance our coverage analysis through statistically significant evidence towards the benefits of using SART to test modified specifications.

Furthermore, the classification used by SART divides test cases in three different categories. That is a construct different from the classification presented by Leung and White [5] that divides regression test cases into five different categories. Even though we may not have an accurate construct to those five classifications, our countermeasure is that the *new-specification* and *retestable* classification are all covered in the *targeted* classification used by SART. A similar problem is reported in [7,15] where they also adapt Leung and White's classification to a finer grained classification to handle peculiarities of high level artefacts.

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 severe construct validity threats since controlling all those assumptions in an experiment can encumber analysis. Our countermeasure was to first do a sanity check with the case study and then gather strong evidence that SART behaves as well as known STCS and also targets full modification coverage (as we observe in our analysis).

5.5.2. External validity threats

Most of our external validity threats are related to the case study. 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. As a countermeasure we avoid stating general results regarding the case study, and limit our main and general contributions (such as SART's safety) to the experiment since we provide statistical and practical significance of results.

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. As a countermeasure we guide our case study analysis based on the test suite's dimensions (size, number of defects, etc.) and the techniques efficiency, avoiding to favour any of them regarding safety and precision.

In our next experiment we intend to focus on test suites exported by tools such as TestLink that allows testers to create test plans for different versions of the SUT. Therefore, we are able to investigate different scenarios and investigate SART's independence from models and aim to more general conclusions regarding SART's safety and robustness in covering modifications.

5.5.3. Conclusion validity threats

As countermeasures to avoid conclusion validity threats, we use non-parametric statistics with consistent constructs (e.g. STCS techniques) as suggested in [20]. In addition, we carefully determine our sample size through a prior power analysis to achieve statistically significance in our results, and run normality tests to check whether a parametric analysis is suitable.

5.5.4. Internal validity threats

The main internal validity threats are related to the operation of the experiment, such as implementation of the techniques and the experimental environment. As countermeasures we thoroughly test the techniques and similarity functions prior to execution. Note that the results are consistent to the observations in [9] that also experiments with different STCS techniques.

6. Related work

Test case selection for regression testing is a widely researched topic in literature, resulting in several proposed techniques, most of them targeting artefacts 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 popular criterion for selection 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 Szermer [24]. The goal was to compare control flow graphs obtained from different versions of a source code and then

[18] Using a larger sample of more varied industrial models.

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 [7,15,25]; 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 component, or change a complex component of the software).

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 to complement the model comparison. In fact, dependence analysis for model-based regression testing is an extensively studied topic [3,7]. Several approaches have been proposed for UML models such as, sequence diagrams, state machines, use cases and class diagrams [1,7,14,15,25]. They all address models differently by analysing information of the same SUT but from different models (e.g. class and sequence diagrams connected to a use case [15]).

In dependency analysis of specification models [1,14], 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 affect the modification, the modified part can be affected by the model, and a side effect can be introduced by the modification. Comparison alone may not be sufficient to ensure safe regression testing in model-based testing, because 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 hand, those techniques tend to be costly in practice, or limited by constraints (e.g. a system's size). Among the selection strategies described above, SART *identifies modifications* and *analyse* similarities between test cases to increase coverage of modification-traversing test cases. But unlike other techniques that rely on model comparison or dependence analysis, SART applies similarity-based test case selection (STCS) to identify modifications based on information from test cases. Our technique does not require model files as input. However, we assume that the test cases contain behavioural information of the SUT, such as basic and alternative flows, instead of just test data.

7. Concluding remarks

This paper presented the similarity approach for regression resting (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 [19] and the artefacts of our evaluation are available online[19] (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 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 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 perform 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 or text information, 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 STCS able to identify modification-traversing test cases.

[19] https://sites.google.com/site/fgonetosite/home/downloads

Acknowledgements

This research project is part of a collaboration between the Federal University of Campina Grande (Brazil) and Chalmers and the University of Gothenburg (Sweden) under the *Science Without Borders* (SWB) [grant number 88881.030428/2013-01, project number 152146]. Also, this work was partially supported by the National Institute of Science and Technology for Software Engineering, funded by CNPq/Brasil [grant number 573964/2008-4]; and by the Knowledge Foundation (KKS) through the project 20130085: Testing of Critical System Characteristics (TOCSYC).

References

[1] B. Korel, L.H. Tahat, B. Vaysburg, Model based regression test reduction using dependence analysis, in: Proceedings of the International Conference on Software Maintenance (ICSM '02), IEEE Computer Society, Washington, DC, USA, 2002, pp. 214–223.
[2] M.J. Harrold, A. Orso, Retesting software during development and maintenance, in: Frontiers of Software Maintenance (FoSM 2008), Beijing, China, 2008, pp. 99–108.
[3] S. Yoo, M. Harman, Regression testing minimization, selection and prioritization: a survey, Softw. Test. Verification Reliab. 22 (2) (2012) 67–120.
[4] G. Fraser, F. Wotawa, Redundancy based test-suite reduction, in: M. Dwyer, A. Lopes (Eds.), Fundamental Approaches to Software Engineering, Lecture Notes in Computer Science, vol. 4422, Springer Berlin Heidelberg, 2007, pp. 291–305.
[5] H.K.N. Leung, L. White, Insights into regression testing, in: Software Maintenance, 1989., Proceedings., Conference on, 1989, pp. 60–69.
[6] G. Rothermel, M.J. Harrold, Analyzing regression test selection techniques, IEEE Trans. Softw. Eng. 22 (1996) 529–551.
[7] E. Fourneret, J. Cantenot, F. Bouquet, B. Legeard, J. Botella, Setgam: generalized technique for regression testing based on uml/ocl models, in: Software Security and Reliability (SERE), 2014 Eighth International Conference on, 2014, pp. 147–156, doi:10.1109/SERE.2014.28.
[8] E.G. Cartaxo, P.D.L. Machado, F.G. de Oliveira Neto, On the use of a similarity function for test case selection in the context of model-based testing, Softw. Test. Verification Reliab. 21 (2) (2011) 75–100.
[9] H. Hemmati, A. Arcuri, L. Briand, Achieving scalable model-based testing through test case diversity, ACM Trans. Softw. Eng. Methodol. 22 (1) (2013) 6:1–6:42.

[10] M. Fahad, A. Nadeem, A survey of UML based regression testing, in: Z. Shi, E. Mercier-Laurent, D. Leake (Eds.), Intelligent Information Processing IV, IFIP Advances in Information and Communication Technology, vol. 288, Springer Boston, 2008, pp. 200–210.
[11] F.G. de Oliveira Neto, R. Feldt, R. Torkar, P.D.L. Machado, Searching for models to evaluate software technology, in: Proceedings of the 1st International Workshop on Combining Modelling and Search-Based Software Engineering, 2013, pp. 12–15.
[12] G. Cabral, A. Sampaio, Formal specification generation from requirement documents, Electron. Notes Theor. Comput. Sci. 195 (2008) 171–188.
[13] C. Jard, T. Jéron, TGV: theory, principles and algorithms: a tool for the automatic synthesis of conformance test cases for non-deterministic reactive systems, Int. J. Softw. Tools Technol. Transfer 7 (4) (2005) 297–315.
[14] Y. Chen, R.L. Probert, H. Ural, Regression test suite reduction using extended dependence analysis, in: SOQUA '07: Fourth international workshop on Software quality assurance, ACM, New York, NY, USA, 2007, pp. 62–69.
[15] L.C. Briand, Y. Labiche, S. He, Automating regression test selection based on UML designs, Inf. Softw. Technol. 51 (1) (2009) 16–30. Special Section - Most Cited Articles in 2002 and Regular Research Papers
[16] M.J. Harrold, R. Gupta, M.L. Soffa, A methodology for controlling the size of a test suite, ACM Trans. Softw. Eng. Methodol. 2 (3) (1993) 270–285.
[17] A. Bertolino, E.G. Cartaxo, P.D.L. Machado, E. Marchetti, J.a.F.S. Ouriques, Test suite reduction in good order: Comparing heuristics from a new viewpoint, in: The 22nd IFIP International Conference on Testing Software and Systems (ICTSS'10), 2010, pp. 13–18.
[18] Testing Techniques in Software Engineering, in: P. Borba, A. Cavalcanti, A. Sampaio, J. Woodcock (Eds.), Springer-Verlag, Berlin, Heidelberg, 2010.
[19] 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 embedded systems, in: Proceedings of the 2008 ACM Symposium on Applied Computing, in: SAC '08, ACM, New York, NY, USA, 2008, pp. 1540–1544.
[20] A. Arcuri, L. Briand, A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering, Softw. Test. Verification Reliab. 24 (3) (2014) 219–250.
[21] R. Jain, The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modeling, John Wiley, 1991.
[22] A. Vargha, H.D. Delaney, A critique and improvement of the CL common language effect size statistics of McGraw and Wong, J. Educ. Behav. Stat. 25 (2) (2000) 101–132, doi:10.3102/10769986025002101.
[23] E.G. Cartaxo, F.G. de Oliveira Neto, P.D.L. Machado, Test case generation by means of UML sequence diagrams and labeled transition systems, in: Proceedings of IEEE International Conference on Systems, Man and Cybernetics, 2007 (SMC'07), 2007, pp. 1292–1297.
[24] J. Laski, W. Szermer, Identification of program modifications and its applications in software maintenance, in: ICSM '92: Proceedings of the Conference on Software Maintenance, IEEE Computer Society, 1992, pp. 282–290.
[25] Q.-u.-a. Farooq, M.Z.Z. Iqbal, Z.I. Malik, M. Riebisch, A model-based regression testing approach for evolving software systems with flexible tool support, in: Proceedings of the 2010 17th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, in: ECBS '10, IEEE Computer Society, Washington, DC, USA, 2010, pp. 41–49.