SOFTWARE TESTING, VERIFICATION AND RELIABILITY
Softw. Test. Verif. Reliab. 2020; **00**:1–17
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/stvr

Assessing test suites of extended finite state machines against model and code based faults

K. El-Fakih*[1] A. Alzaatreh[1], U. C. Turker[2]

[1]*American University of Sharjah, Sharjah, Po. Box 26666, UAE,* [2] *University of Leicester, Leicester, University Rd, Leicester LE1 7RH, UK*

SUMMARY

Tests can be derived from extended finite state machine (EFSM) specifications considering the coverage of single transfer faults, all transitions using a transition tour, all-uses, edge-pair, and prime path with side trip. We provide novel empirical assessments of the effectiveness of these test suites. The first assessment determines for each pair of test suites if there is a difference between the pair in covering EFSM faults of six EFSM specifications. If the difference is found significant, we determine which test suite outperforms the other. The second assessment is similar to the first; yet, it is carried out against code faults of twelve Java implementations of the specifications. Besides two assessments are provided to determine whether test suites have better coverage of certain classes of EFSM (or code) faults than others. The evaluation uses proper data transformation of mutation scores and p-value adjustments for controlling Type I error due to multiple tests. Furthermore, we show that subsuming mutants have an impact on mutation scores of both EFSM and code faults; and accordingly, we use a score that removes them in order not to invalidate the obtained results. **The assessments show that all-uses tests were outperformed by all other tests; transition tours outperformed both edge-pair and prime path with side trips; and single transfer fault tests outperformed all other test suites. Similar results are obtained over the considered EFSM and code fault domains and there were no significant differences between the test suites coverage of different classes of EFSM and code faults.**

Received . . .

KEY WORDS: Model-based testing, mutation testing, extended finite state machines, empirical assessment, mutation scores.

1. INTRODUCTION

Testing is an indispensable yet expensive part of software development. One promising approach to reduce this cost is to use model-based testing techniques and tools. Accordingly, many techniques are provided for test selection of systems modeled as extended finite state machines (EFSMs). These machines have powerful modeling capabilities and well-defined semantics based on extending the traditional (Mealy) FSM model with variables, predicates, and update statements. In particular, an EFSM transition is labeled by an input/output pair of possibly parameterized interactions and the transition can be guarded by a predicate that must hold for the transition to be executed. Upon its execution, new values can be assigned to variables based on the transition update statements. Due to these properties, EFSMs are used for functional testing of many systems including communication networks, embedded components, web services, object oriented systems, etc. For some related work the reader may refer to [9, 10, 14, 19, 21, 40, 41, 53, 55, 62, 63, 64].

A *trace* of an EFSM specification M is a sequence of (concrete) input/output interactions starting from the initial configuration of M. A *configuration* simply consists of a pair; a state and a vector

*Correspondence to: American University of Sharjah, Sharjah, PO Box 26666, UAE

Prepared using stvrauth.cls *[Version: 2010/05/13 v2.00]*

of current valuations of variables. A *test case* is finite trace of M and such a trace is *feasible or executable* by definition. A test suite is a finite set of feasible test cases.

A wide range of test selection criteria can be used for deriving tests from a given EFSM specification M. For instance, tests can be derived from the (flow-)graph representations of M using the traditional all-uses [31], edge-pair, and prime path with side trip [5] criteria. In addition, a known way to derive tests from M is to enumerate all single transfer fault EFSM mutants of M, and then derive a test suite (called single transfer faults test suite) that has the capability of detecting/distinguishing each mutant that does not have the same behavior as M. **An EFSM mutant M' of M has a *single transfer fault* if M' is an exact copy of M; except for one transition t of M with the ending state s, t in M' has an ending state s' that is different from s.** Another way of test derivation is to traverse all the edges of M using a single test case called a *transition tour*.

Considering the diversity of EFSM based test selection criteria, we need to investigate the effectiveness of these test suites in covering faults in order to use the most appropriate one(s). This is done using mutation testing [2, 5, 36] where seeded faults, derived according to some established mutation operators, are inserted into the implementation of a given EFSM specification producing related (code) mutants. However, the assessment can also be done using EFSM based mutants (faults) of the specification. These mutants are derived with respect to traditional EFSM based mutation operators which are mostly based on the EFSM transitions; such as considering transitions output parameter, transfer, and assignment update faults. In both cases, the assessment is carried out using the mutation scores of the considered test suites. Traditionally, the mutation score MS of a certain test suite, with respect to a given collection of mutants, is computed based on the number of mutants killed by the suite divided by the number of all derived mutants minus the number of alive mutants that cannot be killed by any test suite. However, in a recent work, Papadakis et al. [52] proposed an enhanced mutation assessment score, denoted hereafter as MS^*. The proposed mutation score is based on removing subsuming mutants before computing the mutation score; as these mutants inflate the mutation scores and thus may affect the validity of the assessment studies. Accordingly, in this paper, we also use the mutation score MS^*. It is worth mentioning that recent and established research [6, 39] demonstrate a strong relationship between code mutants and real faults. This indicates that results obtained using code mutation test assessment, as done in this paper, may apply to findings that would pertain to real faults in real systems [6, 39].

In this paper, we consider six realistic EFSM specifications and related single transfer fault, all-uses, edge-pair, prime path with side trip, and transition tour test suites. Then, we conduct experiments with the aims and contributions illustrated below.

- **Considering the diversity of test selection from EFSMs; our first aim is to empirically assess the effectiveness of the test suites in terms of their coverage of EFSM faults. To this end, we provide an empirical assessment that determines for each pair of considered test suites whether there is a significant difference between the pair in covering the EFSM faults of six real EFSM examples. If the difference is significant, we determine which test suite outperforms (has better coverage than) the other.**
- **The second aim of this work is to empirically assess the test suites; however, with respect to their coverage of the code faults of twelve Java implementations of the considered EFSM examples**
- **The third (fourth) assessment provided in this paper aims at determining whether the test suites have better coverage of certain classes of EFSM (code) faults than others. This will tell us whether the tests derived from EFSMs are biased against detecting certain classes of faults or not.**
- **Another aim of this paper is to study the influence of subsuming mutants on mutation testing experiments. In [52] subsuming code mutants are shown to inflate the mutation scores; and thus, they should be removed in order not to invalidate the outcomes of related studies. Here, we provide a simple correlation analysis between the traditional MS and the enhanced mutation MS^* scores verifying the same question raised in [52]; however, over the EFSM faults of the considered specifications and the code faults of the**

DOI: 10.1002/stvr

corresponding Java implementations. Accordingly; we use the mutation score MS^* in our assessments.

- **The last contribution of the paper is the selection of appropriate statistical evaluation methods for the considered assessments. When multiple tests are conducted, the outcome without adjustment of the p-values is questionable. For instance, each of the first two assessments handles 10 pairwise comparisons as we consider five different test suites; and thus, if done independently, there will be a higher risk of Type I error (detecting a false significance) than the intended significant level of 5% [59]. To appropriately handle this situation; the statistical evaluation first tests the overall significance after using a proper data transformation of the mutation scores so that the ends of the scale of the distribution is expanded. If the overall significance is detected, pairwise comparisons are conducted to find the source of the difference. Then the resulting p-values are adjusted using Bejamini and Hochberg [8] method to control the probability of Type I error due to multiple simultaneous tests. Therefore, conclusions are made based on the adjusted p-values (called q-values) and not the original p-values.**

Related Work: **This work mostly relates to the previous work reported in [23] and [24], respectively. In [23] three (out of the five) EFSM based test suites considered in this paper are sorted (from best to worst) with respect to their coverage of code faults, and in [24] the test suites considered in this paper are sorted with respect to their coverage of EFSM faults. The mutation score used in these studies was the traditional mutation score. In this paper, we provide two simple correlation studies that show, as in other related work [52], that subsuming mutants affect mutation scores. Accordingly, the more appropriate mutation score MS^* is used in the assessments provided in this paper. However, more importantly, in this paper, we empirically assess the effectiveness of the test suites. To this end, proper research questions are elaborated and a proper statistical evaluation method is used allowing us to provide sound conclusions. No empirical assessments are provide in [23, 24]. Besides, here, we also provide two novel assessments showing that there is no significant differences between the test suites coverage of different classes of EFSM and code faults.**

In general, for a good survey and information about mutation testing, the reader may refer to Ammann and Offutt [5], Jia and Harman [36], and Mathur [2]. The reader may also refer to [1,3,7,17,33,46] for summaries on model-based mutation testing; especially for specification written in UML [3, 11, 42, 45], statecharts [26], SDL [44], and (Mealy) FSMs [20, 54]. Other line of research focuses on assessing code based testing and mutation testing against code faults [6, 15, 27, 28, 29, 30, 35, 47, 48, 50]. Also there has been some work on the derivation of tests and distinguishing tests for EFSMs [10, 14, 21, 25, 40, 41, 53, 63]. Thus, our work complements this previous research; however, we focus on a different specification model; namely, the EFSM model, where we assess related test suites against both EFSM and code faults.

This paper is organized as follows. Preliminaries are introduced in Section 2. Section 3 includes the provided empirical studies and Section 5 includes a summary of obtained results. Section 5 concludes the paper.

2. PRELIMINARIES

2.1. *Extended finite state machines*

A (Mealy) finite state machine (FSM) is an initialized machine with finite number of inputs X, outputs Y, states S with the initial state s_0, and transitions. A transition t has the form $t = (s, x, y, s')$ and it means if the machine is in state s, upon receiving an input x, it produces the output y and moves to state s'.

The EFSM model extends the FSM model with variables V, update statements, predicates, and (possibly parameterized) inputs and outputs. The reader may refer to [10, 21, 24, 53] for formal

DOI: 10.1002/stvr

definitions of such machines. A transition t has the form $t = (s, x(p), [G], up, op, y(q), s')$, where s and s' are the starting and ending states of t, $x \in X$ ($y \in Y$) is an input (output) interaction with the parameter p (q), $[G]$ is the enabling predicate of t which depends on the current values of state variables and input parameter value ρ, and up is are concurrent update statements which define new values for certain variables in V as a function of the current values of variables and the parameter p, op is an output parameter function which updates the output parameters of t according to values of variables and q. For simplicity of presentation, we use interactions with single parameters. The meaning of t is the following: If M is in state s, then M may make a transition to state s' by receiving the input x with parameter value ρ if $[G]$ holds True for the current values of variables $\mathbf{v}$ and ρ. If t is executed, the values of variables are updated according to the update in up producing $\mathbf{v}'$, the value of q will be updated according to op, and afterward the output y is produced carrying the value ϱ of q; i.e. $y(\varrho)$. Such an execution realizes the corresponding concrete (Mealy) FSM transition $((s, \mathbf{v}), x(\rho), y(\varrho), \mathbf{v}')$, with the concrete input/output pair $x(\rho)/y(\varrho)$.

Considering a sequence of consecutive transitions of M, we obtain a corresponding (*feasible* or *executable*) trace over concrete input/output pairs of these transitions.

A *test case* derived from an EFSM is a finite length trace of the machine. A finite set of such test cases is a *test suite*. By definition, such test cases (suites) are executable. *Test case length* equals the total number of inputs of a test case and *Test suite length* is the total length of all test cases of a test suite.

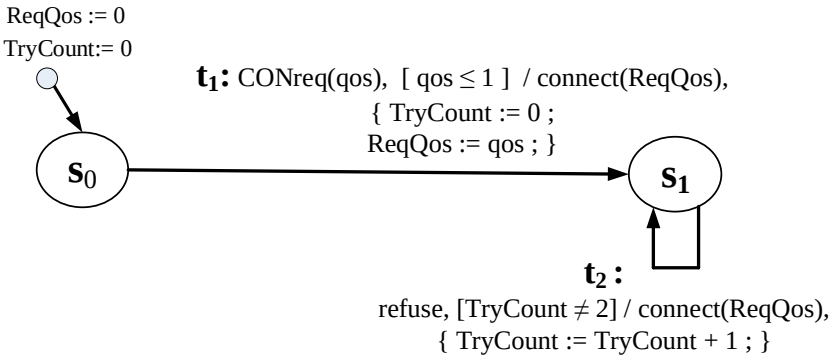


Figure 1: Part of the SCP protocol.

As a simple example, we consider an execution of two consecutive transitions $t_1.t_2$ of the SCP protocol [16]. Part of this protocol is shown in Fig. 1. The protocol has two integer variables $TryCount$ and $ReqQos$ with the possible values 0 or 1. Transition t_1 has the predicate $[qos \leq 1]$, update statements $\{TryCount := 0; ReqQos := qos\}$, parameterized input $CONreq$ with integer parameter qos which can have values 0 or 1, and parameterized output $connect$ with integer output parameter carrying the value of $ReqQos$. Transition t_2 has guard $[TryCount \neq 2]$, update $TryCount := TryCount + 1$, input $refuse$, and the same output as t_1. The initial state of the SCP machine is s_0 and the initial values of $TryCount$ and $ReqQos$ are zeros. Assume the machine receives the input $CONreq(1)$, i.e., input $CONreq$ with parameter $qos = 1$. Then t_1 can be executed as its predicate $[qos \leq 1]$ is satisfied, then upon the execution of t_1, the update statements are executed setting $TryCount := 0$ and $ReQos := 0$, and the output $connect(0)$ is then produced, and the machine moves to state s_1. In fact the machine has the corresponding concrete transition $((s_0, (TryCount = 0, ReqQos = 0)), CONreq(1), connect(0), (s_1, (TryCount = 0, ReqQos = 0))$. Afterwards, if the machine receives the input $refuse$, then t_2 can be executed as $[TryCount \neq 2]$ is satisfied. Then, upon the execution t_2, $TryCount := TryCount + 1$ is executed, the output $connect(0)$ is produced, and the machine moves to state s_1. In fact, it has the corresponding concrete transition $((s_1, (TryCount =$

DOI: 10.1002/stvr

$0, ReqQos = 0)), refuse, connect(0), (s_1, (TryCount = 2, ReqQos = 0))$. Thus, the above execution has the corresponding trace $CONreq(0)/connect(0).refuse/connect(0)$.

2.2. EFSM mutants

For a given EFSM specification M, its corresponding EFSM mutants $\Im^{EFSM}$ are derived using the traditional EFSM mutation operators $OP^{EFSM} = \{$ TF, OPF, AF $\}$ listed below. For every op_i in OP^{EFSM}, $\Im^{op_i}$ denotes the collection of EFSM mutants (class of faults) derived from M using this operator, and $\Im^{EFSM}$ denotes the collections of mutants derived using all operators in OP^{EFSM}. Table below I shows how an operator can be applied to a transition $t = (s, x(p), [G], up, op, y(q), s')$ of M, producing the corresponding EFSM mutant. The reader may refer to [9,21,24,62] for detailed description of these mutants.

Table I. **EFSM Mutation Operators**

Operator		Description/Example(s)
TF	(Single) Transfer Fault	The ending state s' of transition t is replaced by state $s'' \neq s'$
OPF	Output Parameter Fault	The output parameter function op of t is replaced by $op' \neq op$ e.g., op defined over a context variable (or a constant) is replaced by another context variable or a constant in op'.
AF	(Single) Assignment Fault	The update up of transition t replaced by $up' \neq up$, e.g., update statement of another transition is added into up (assignment insertion) e.g., update statement in up is removed (assignment deletion) e.g., Right-Hand-Side (RHS) of up is changed; as examples: a variable or a constant added into the RHS of up; a variable or a constant in the RHS of up is deleted or replaced by another variable or constant.

2.3. Code mutants and mutation operators

For a given Java implementation, its corresponding code mutants $\Im^{Code}$ are derived using the well-studied mutation operators $OP^{Code} = \{ COR, ROR, EVR, LVR, STD, AOR \}$ listed below [39]. For every op_i in OP^{Code}, $\Im^{op_i}$ denotes the collection of (code) mutants derived from the given implementation using this operator, and $\Im^{Code}$ denotes the collections of mutants derived using all the operators in OP^{Code}. As usual, 1-Order code mutants are considered to alleviate problems related to the coupling effect of using N-order mutants, when $N > 1$.

Table II. Java Mutation Operators [37]

Operator		Description/Example(s)
COR	Conditional Operator Replacement	e.g. $a\|\|b$ replaced by $a\&\&b$
ROR	Relational Operator Replacement	e.g. $a == b$ replaced with $a >= b$
EVR	Expression Value Replacement	An expression with a default value, e.g. int $a = b$ with int $a = 0$
LVR	Literal Value Replacement	Numerical literal is replaced by positive number, negative number, or zero.
STD	(single) Statement Deletion	Deletes an assignment, break, or continue statement
AOR	Arithmetic Operator Replacement	e.g. $a + b$ replaced with $a - b$

DOI: 10.1002/stvr

2.4. Test Suites

In this paper we consider the following EFSM based test suites. These test suites are derived, as will be illustrated in Section 3.2, mostly, using the criteria and tools described in [5, 9, 10, 18, 23, 24, 25, 31, 53, 61, 64].

(a) Single transfer fault test suite (STF)- A test case *distinguishes/kills* an EFSM mutant M' from a given EFSM specification M if the output sequences of these machines, with respect to the input sequence of the test case, are different. A test suite that kills each single transfer fault mutant, that has a different behavior than M, is called a single transfer fault test suite (STF).

(b) Transition tour (TT)- A TT of M is a single test case that traverses all transitions of M.

(c) All-$Uses$- Given a flow-graph representation of M, annotated with definitions and uses of variables and parametrized inputs, derived as illustrated in [10, 64]. An All-$Usess$ test suite can be derived from the obtained annotated flow-graph using the traditional data-flow all-uses criterion [31].

(d) EP and $PPST$- These test suites can be derived from the graph representation of an EFSM specification M using the criteria illustrated in [5]. An *edge-pair test suite* EP is a test suite that covers each executable path of length up to 2 of a given graph. More precisely, edge-pair coverage requires covering each pair of consecutive edges, or a path of length 2 of the given graph. Further, a path from node n_i to node n_j in the graph is *simple* if no node appears more than once in the path, with the exception that the first and last nodes may be identical. A path from node n_i to node n_j is a *prime path* if it is a simple path and it does not appear as a proper sub-path of any other simple path. A *prime path with side trip* is a path p that tours the prime path q such that every edge in q is also in p in the same order. In this paper, we consider prime path with side trip test suites, denoted by $PPST$

2.5. Mutation scores

Traditionally, the mutation score of a test suite TS with respect to a collection of mutants $\Im$, is computed as the following should be $MS = Killed/(All - Alive) * 100$ where, All denotes the total number of mutants in $\Im$, $Killed$ denotes the number of those killed by TS, and $Alive$ denotes the number of mutants in $\Im$ that cannot be killed by any test suite as they have the same behavior as the original code (machine).

Also please note that, in order not to inflate the score due to subsuming mutants, the score, denoted MS^*, is computed after removing all subsuming mutants killed by TS from $\Im$ [52]. Note that one mutant *subsumes* another if at least one test case kills the first, and every test case that kills the first also kills the second [4]. In Section 3.2, we describe in detail the methods and tools used for determining $Alive$ for code and EFSM mutants, respectively.

3. EMPIRICAL STUDIES

3.1. Research questions

We consider five different types of EFSM based test suites; namely, the test suites $Suites^{EFSM} = \{$ All-$Usess$, EP, STF, $PPST$, TT $\}$.

The objective of the first assessment (**RQ1**) considered in the paper is to compare, based on MS^* mutation scores, the *effectiveness or performance* of each pair of considered test suites (TS_i, TS_j) in $Suites^{EFSM}$ against collections of EFSM mutants (faults) of the considered EFSM examples; and the objective of the second assessment (**RQ2**) is to compare the effectiveness against code mutants (faults) of the Java implementations of these examples.

More precisely, for every pair (TS_i, TS_j) in $Suites^{EFSM}$, we consider the following two hypotheses:

H_0: There is no difference of the average mutation scores between TS_i and TS_j.

H_1: There is a difference of the average mutation scores between TS_i and TS_j.

In fact, we assess the above hypotheses against EFSM faults using the following research question:

DOI: 10.1002/stvr

RQ1: We test the above hypotheses for each pair of test suites (TS_i, TS_j) in $Suites^{EFSM}$ considering the collections of EFSM mutants $\Im^{EFSM}$ derived from the six considered EFSM examples and their corresponding mutation scores.
In addition, we assess the hypotheses against code faults using the following research question:

RQ2: We test the above hypotheses for each pair of test suites (TS_i, TS_j) in $Suites^{EFSM}$ considering the collections of code mutants $\Im^{Code}$ derived from the twelve considered Java implementations (of the EFSM examples) and their corresponding mutations scores.

Another two aims of the this work is to assess if the test suites have better coverage of certain classes of EFSM or code faults than others. Accordingly, in the third assessment (**RQ3**), we to determine for each test suite TS in $Suites^{EFSM}$ and for every selected pair of the selected classes of EFSM faults $\Im^{op_i}$ and $\Im^{op_j}$, obtained using the corresponding operators op_i and op_j, if there is a significant difference in the coverage of TS of the mutants in these classes. The fourth assessment (**RQ4**) is similar to the third; yet it is carried out over the different pairs of selected classes of the Java faults of the considered implementations.

Thus, we consider testing the following two hypotheses for each TS in $Suite^{EFSM}$:

H'_0: There is no difference of the average mutation scores between $\Im^{op_i}$ and $\Im^{op_j}$.

H'_1: There is a difference of the average mutation scores between $\Im^{op_i}$ and $\Im^{op_j}$.

In fact, we assess the above hypotheses against different classes of EFSM (code) of faults using the following two research questions:

RQ3: We test H'_0 and H'_1 for each pair of operators (op_i, op_j) in OP'^{EFSM} over each test suite TS in $Suite^{EFSM}$.

This assessment considers the different classes of EFSM mutants $OP'^{EFSM} = \{ TF, AF \}$. Note the class containing single output parameter faults mutants $\Im^{OPF}$ is not included this assessment as most of the considered examples did not have parameterized outputs; i.e. corresponding collections $\Im^{OPF}$ are empty.

RQ4: We test H'_0 and H'_1 for each pair of operators (op_i, op_j) in OP'^{Code} over each test suite TS in $Suite^{EFSM}$.

This assessment considers the following classes of Java mutants $OP'^{Code} = \{ COR, ROR, LVR, STD \}$. Note only mutants of these classes are considered in this assessment; as the vast majority of derived mutants come from these classes. In other words, the AOR and EVR operators produced significantly less mutants than other operators.

In this paper, we also study the correlation between the traditional mutation score MS and the score MS^* computed after removing subsuming mutants. As in [52], if there is a strong correlation, we infer that the influence of subsumed mutants on the score is minor; otherwise, the effects may be distorting. More precisely, we investigate the following two questions:

RQ5: Does the mutation score MS of considered EFSM test suites computed based on EFSM mutants have high correlation with the subsuming mutation score MS^*.

RQ6: Does the mutation score MS of considered EFSM test suites computed based on code mutants have high correlation with the subsuming mutation score MS^*.

3.2. Assessment methodology

Here we describe the assessment method used in this paper.

Step-1: **Application examples**: We consider six well-known realistic EFSM specification examples; namely, the Trivial File Transfer Protocol (TFTP) [56]; Post Office Protocol V.3 (POP3) [57]; Initiator [34]; Responder [34]; SCP [16], and the CD player [58].

Step-2: **Java Implementations:** For each considered example, corresponding Java implementations are developed, by different software engineers, based on the EFSM specification and its textual description, under the following coding rules. State variables cannot be explicitly or implicitly introduced in an implementation; for instance, no state variables nor flags or labels indicating states can be used. In addition, names of (parameterized) inputs and outputs of the EFSM specification should be preserved in a code implementation. Each implementation should

DOI: 10.1002/stvr

be implemented as one function that inputs a string separated by a delimiter ”,” representing an input sequence to the function and returns as an output a string representing the output response of the implementation to the input sequence. A Reader/Writer class is used in all implementations that handles reading/writing the input and the output strings in order to separate reading and writing outputs from the function that implements the specification and thus, code mutants are only derived from the function that implements the specification. We note each Java implementation is thoroughly tested using all considered test suites. In total twelve implementations are considered in the assessments.

Step-3: **Derivation of Mutants**: **For each EFSM example, corresponding EFSM mutants with single transfer and single output parameter faults are derived using a software tool that we have developed for this purpose [24]. However, assignment faults are injected by hand, one after the other, into the specification and their corresponding mutants are then saved by the developed tool [24]. Furthermore, for each Java implementation, the Major tool framework [37] was used to derive related Java mutants considering the mutation operators given in Table II. We note several studies [39, 43], considering the widely-used Defects4J [38] data set, show that majority of real faults are coupled to mutants generated by the used Major tool framework.**

Step-4: **Derivation of Test Suites**: **We use the same test suites considered in [24]. However, we recall here how these suites are derived for illustration purposes. Deriving a STF test suite of an EFSM specification is carried out by first deriving all single transfer fault mutants of the specification using a tool that we have developed for this purpose [24]. Then for each considered EFSM mutant, the Plavis/FSM tool [61] is used to derive and add to the test suite a test case that distinguishes the mutant from the corresponding specification if needed (i.e., if the test suite does not already have a test case that kills the mutant). We note that a derived STF test suite is of optimal or near-optimal length as the tool [61] derives shortest length distinguishing tests. We note that methods that describe procedures for deriving distinguishing tests for two EFSMs are illustrated in detail in [10, 25, 53]. In fact, our tool determines if an EFSM mutant is distinguishable from an EFSM specification or not as follows: First the tool simulates the behavior of the EFSMs and produces the corresponding FSMs as illustrated in [22]. Then, the Plavis/FSM tool [61] is used to determine if the two FSMs are distinguishable or not. We note that in general, when the domains of variables are not finite, an EFSM may not have a corresponding FSM; however, all the EFSM examples considered in this paper have corresponding FSMs and thus we could determine the distinguishability of two EFSMs using the corresponding FSMs.**

A TT test suite is derived (by hand) as a selected path of an EFSM specification, and then the feasibility of the path is checked and a corresponding test case is derived. Again, we derive optimal or near-optimal length TT test suites. Deriving All-Uses test suites is done as follows: For every EFSM specification, a corresponding flow-graph representation, annotated with definitions and uses of variables, is derived (by hand), and then corresponding All-Uses test suite (set of paths) is derived from the obtained flow-graph exactly as described in previous related work [64]. EP and PPST are derived with the help of the graph coverage web application tool [5]. We note that we consider executable test cases in this paper. That is, for every derived path in the flow-graph (graph) we check that if it corresponds to an executable test case using the corresponding transitions in the EFSM and making sure to select appropriate values for input parameters, execute update statements of the transitions, and determine the reached configurations. The obtained tests, traces over the EFSM specification, are transformed into the corresponding JUnit tests using a simple procedure that we have developed for this purpose. For instance, the trace $CONreq(0)/connect(0).refuse/connect(0)$ of the EFSM in Fig. 1, provided in Section 2.1, is written in JUnit as follows:

```
import junit.framework.TestCase;
public class testSCP extends TestCase {
public SCP tester; public void setUp()  tester = new SCP(); }
public void test1() {
```


DOI: 10.1002/stvr

assertEquals(”connect(0),connect(0),”, tester.scp(”CONreq(0),refuse,”)); } }

Step-5: **Computing Mutation Scores**: **Determining the mutation score MS of an EFSM test suite TS with respect to a selected collection of EFSM faults $\Im^{EFSM}$ is done as follows: First, the Plavis/FSM tool [61] is used to determine *Alive* mutants as described above; i.e. mutants in $\Im^{EFSM}$ that are indistinguishable from the considered EFSM specification. Then, by running the test cases of TS against each remaining (non-alive) mutant, we determine which mutants are killed by TS; and accordingly determine related MS. Also, for each TS, we derive a related two-dimension *kill-matrix* that indicates for each test case which (non-alive) mutants are killed by the test case. This is needed to determine subsuming mutants and compute MS^* as illustrated below.**

Determining the mutation score MS of a test suite TS against a collection $\Im^{Code}$ of code mutants of a given Java implementation is done as follows: First to determine *Alive* mutants, all the test suite considered in this paper derived for an EFSM example M are combined into a single test suite; and then the Major tool [37] is used to derive the corresponding mutants, execute the tests on these mutants, and afterward; produce a related *kill-matrix*. Then, *Alive* is determined as the number of mutants that are not killed by any test case of the combined test suites. For code, the distinguishability (or equivalence) problem is in general undecidable [13]. Accordingly, we follow the practice of declaring a mutant alive if it is not killed by a collection of a large number of test suites. This practice is regarded as acceptable as the focus is on the differences between the different test selection criteria [6]. Then, after determining $Alive$, of each considered TS, we run the Major tool again on the considered mutants, determine the number of killed mutants; and then compute the corresponding MS accordingly. We also determine for each test suite and collection of (EFSM or code) mutants $\Im$ the corresponding mutation score MS^* exactly as described above; however, after eliminating subsuming mutants from $\Im$. Determining subsuming mutants is done based on the *kill-matrix* of the test suite.

3.3. Statistical evaluation

In this subsection, we outline the statistical methods used in order to answer the considered research questions followed by some details on each method.

*Outline of the statistical analysis used for **RQ1-RQ4**:*

i. Data transformation.
ii. Testing the overall difference of the mutation scores averages.
iii. If overall significant difference is detected in (ii), then (a) post hoc tests of pairwise comparisons (if needed) are conducted. Afterwards, (b) the resulting p-values of the multiple tests are adjusted to control Type I error.

(i). Data transformation: In order to test the research hypotheses in Section 3.1, for all research questions, we apply the logit transformation to all mutation scores for the considered samples. The logit transformation is defined as $logit(MutationScore) = \log\left(MutationScore/(1 - MutationScore)\right)$, and is often used to transform percentage data so that the ends of the scale of the distribution is expanded. And therefore, it shows a better picture of the difference in proportions. Also, the logit transformation may help making the distribution more symmetrical. For more details, one is referred to [12].

(ii). Testing the overall significance: For **RQ1** and **RQ2**, we test the overall significance among the considered test suites. That it is, we test the following hypotheses:

H_0'' : There is no difference of the average mutations scores of TS_i for all $i = 1, 2, \ldots, 5$.

H_1'': There is at least one difference of the average mutations scores between TS_i and TS_j for some $i \neq j = 1, 2, \ldots, 5$.

Where,

- For **RQ1**, we test the hypotheses using the collections $\Im^{EFSM}$ of EFSM mutants derived from the considered EFSM examples.

Prepared using stvrauth.cls
DOI: 10.1002/stvr

- for **RQ2**, we test the hypotheses using the collections $\Im^{Code}$ of code mutants derived from the considered Java implementations.

For **RQ3** and **RQ4**, for every test suite, we test the overall significance among the considered mutants categories OP. That is, we test the following hypotheses:

H_0''' : There is no difference of the average mutations scores of op_i for all $i = 1, 2, \ldots, k$, $k = |OP|$.

H_1''': There is at least one difference of the average mutations scores between op_i and op_j for some $i \neq j = 1, 2, \ldots, k$, $k = |OP|$.

Where,

- For **RQ3**, we test the above hypotheses (for each test suite) per the collections of EFSM mutants derived using each op_i in $OP = OP'^{EFSM}$ and their corresponding mutation scores over all considered examples.
- For **RQ4**, we test the above hypotheses (for each test suite) per the collections of code mutants derived using each op_i in $OP = OP'^{Code}$ and their corresponding mutation scores over all considered Java implementations.

For each of the research questions **RQ1-RQ4**, we used the repeated measure Analysis of Variance (rANOVA) to test the above hypothesises (H_0'' and H_0''') with one exception, **RQ3**. Since only two groups (classes) are compared in **RQ3**, the paired t-test is used instead of rANOVA. If the resulting p-value from rANOVA is greater than or equals 5%, we assume there is no overall significant difference of the mutation score average among the comparison groups and therefore, we assume all perform equally. Whereas, a p-value of less than 5% indicates an overall significant difference. For **RQ1**,**RQ2** and **RQ4**, the result from rANOVA was further verified using the nonparameteric Friedman test [32] (Wilcoxon signed-rank test for **RQ3**). The Friedman test is a nonparametric test analogue to the parametric rANOVA test and is often used when some assumptions of rANOVA are violated (such as normality assumption). rANOVA requires the normality assumption of the sampling distributions. The Shapiro-Wilk normality test [60] was used to check the normality assumption in each case. Quantile-Quantile normality plot (Q-Q plot) was also used to verify the results from Shapiro-Wilk test. In Shapiro-Wilk test, the null hypothesis assumes normal distribution and a p-value of greater than 0.05 supports the normality assumption. During the rANOVA computation, the covariance structure of the residuals is checked and is taken into account.

(iii)- Pairwise comparisons and controlling Type I error: Since overall difference was not found significant for both **RQ3** and **RQ4** (see section 3.4), pairwise comparisons was only applied to **RQ1** and **RQ2**. A paired t-test is used to compare the logit transformed data for each paired sample (TS_i, TS_j). Then, due to large number of comparisons, it is important to control the false discovery rate (controlling Type I error). The Benjamini and Hochberg [8] False Discovery Rate adjustment is one of the popular methods in this context. The Benjamini and Hochberg q-value represents the expected proportion of false positives among all comparisons that called significant. In this paper, q-value of less than 0.05 is deemed significant.

*Outline of the statistical analysis used for **RQ5 and RQ6**:*

For **RQ5** and **RQ6**, we conducted a correlation analysis between the traditional mutation score MS and the score MS^* computed after removing subsuming mutants. If a strong correlation is detected, we infer that the influence of subsumed mutants on the score is minor [52]; otherwise, the effects may be distorting. **A correlation value of 0.7 and above is considered high ([49], [51]).**

Pearson and Spearman correlation coefficients are computed between MS and MS^*. The strength of the association is measured in terms of the absolute value of the correlation coefficient. The strength lies between 0 and 1. **Strong correlation of 0.7 and above suggests that MS and MS^* have similar trend. Weak correlation, however, suggests that MS^* should be used instead of MS.**

DOI: 10.1002/stvr

3.4. Results

3.4.1. ***RQ1*** *results considering EFSM faults:* As described in Section 3.3, the normality assumption was first tested for all EFSM based on the logit transformed mutation scores. In all cases, the normality assumption can be assumed (Shapiro-Wilk's p-value > 0.05). The result from Shapiro-Wilk test was further verified using Q-Q plot (see Figure 2). Then, rANOVA test was used to test the overall difference among the mutation scores. If significant difference was found (rANOVA p-value < 0.05), pairwise comparisons using paired t-test are followed. The resulting p-values from all pairwise comparisons are then adjusted using Benjamini and Hochberg False Discovery Rate method. In this paper, q-value of less than 0.05 is deemed significant.

Over EFSM based mutants, single transfer fault test suite clearly outperforms all other test suites. It kills nearly 100% of all mutants in $\Im^{EFSM}$. Thus, including it in the analysis, distorts the results of rANOVA. For this reason, it was assumed to outperform all other test suites (we verified this assumption by running paired t-test between STF and all other test suites) and it was not considered in rANOVA test.

rANOVA test result showed an overall significant difference of the average mutation scores among the test suites (p-value < 0.002). For further verification, Friedman test was also conducted and the test result showed significant difference (p-value < 0.0007). For the multiple comparison, paired t-test was used to compare each pairs of the test suites. The results of the p-value and the q-value (adjusted p-value using Benjamini and Hochberg False Discovery Rate method) are depicted in Table III. The performance direction was based on the sample mean difference of the mutation scores between the pair test suites. It is worth mentioning that the results in Table III were further verified using the nonparameteric Wilcoxon signed-rank test. The significant test suites were identical. **The boxplots of the logit MS^* are depicted in Figure 2**. The Figure supports the results in Table III.

Table III. p-values and q-values (adjusted p-values) for pairwise comparisons of EFSM faults mutation scores (**RQ1**).

Comparison Test Suites		p-value	q-value	Performance Direction
$All\text{-}Usess$	EP	0.0008	0.0020*	$\prec$
	$PPST$	0.0340	0.0408*	$\prec$
	TT	0.0000	0.0002*	$\prec$
EP	$PPST$	0.3020	0.3020	-
	TT	0.0050	0.0075*	$\prec$
$PPST$	TT	0.0010	0.0020*	$\prec$

* indicates significant difference, $TS \prec TS'$ means TS' outperforms TS.

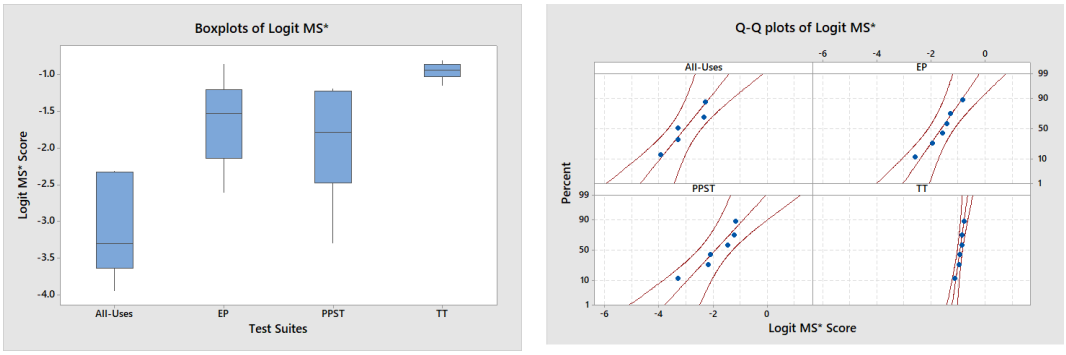


Figure 2. **Boxplots and Q-Q plots of Logit MS* for the EFSM faults mutation (RQ1).**

DOI: 10.1002/stvr

3.4.2. ***RQ2*** *results considering code faults:* The normality assumption was first tested for all code based on the logit transformed mutation scores. In all cases, the normality assumption can be assumed (Shapiro-Wilk'sp-value¿0.05). The result from Shapiro-Wilk test was further verified using Q-Q plot (see Figure 3).Over code based mutants, rANOVA test showed an overall significant difference of the average mutation scores between the test suites (p-value < 0.0001). Friedman test was also conducted and showed significance result (p-value < 0.0001). For the multiple comparison, paired t-test was used to compare each pairs of the test suites. The results of the p-value and the q-value are depicted in Table IV. The results in Table IV were further verified using the Wilcoxon signed-rank test. The significant test suites were identical. **In Figure 3, Boxplots for the logit $MS*$ are plotted for all test suites. The Figure supports the results in Table IV**.

Table IV. p-values and q-values (adjusted p-values) for pairwise comparisons of code faults mutation scores (**RQ2**)

Comparison Test Suites		p-value	q-value	Performance Direction
$All\text{-}Use$	EP	0.0000	0.0003*	$\prec$
	$PPST$	0.0003	0.0007*	$\prec$
	STF	0.0001	0.0003*	$\prec$
	TT	0.0001	0.0003*	$\prec$
EP	$PPST$	0.2280	0.2280	-
	STF	0.0070	0.0140*	$\prec$
	TT	0.0120	0.0200*	$\prec$
$PPST$	STF	0.0270	0.0338*	$\prec$
	TT	0.0350	0.0389*	$\prec$
STF	TT	0.0250	0.0338*	$\succ$

* indicates significant difference, $TS \prec TS'$ means TS' outperforms TS.

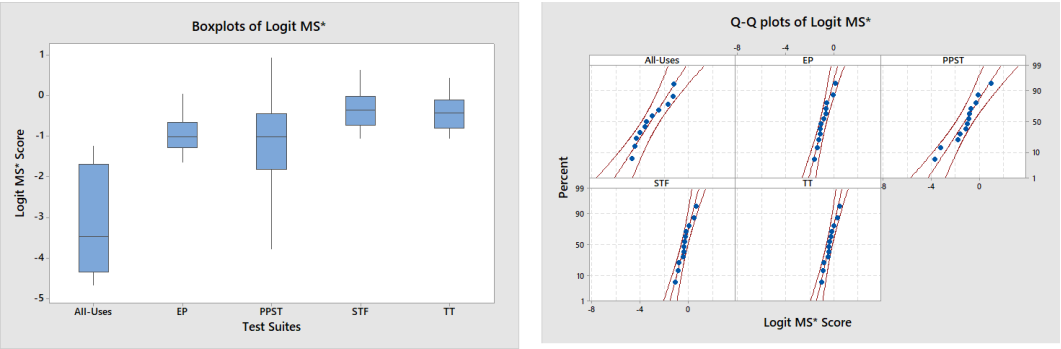


Figure 3. **Boxplots and Q-Q plots of Logit MS* for the code faults mutation (RQ2).**

3.4.3. ***RQ3*** *results over EFSM operators:* For each test suite, paired t-test was used to test the difference of the average mutation scores between the two operators; TF and AF. In all cases, the difference was not found significant (p-value > 0.05). Similar results were also obtained using Wilcoxon signed-rank test.

3.4.4. ***RQ4*** *results over code operators:* For each test suite, rANOVA test was conducted to test the overall difference among the four operators; COR, ROR, LVR and STD. In all cases, the overall difference was not found significant (p-value > 0.05). The results were also verified using Friedman test.

DOI: 10.1002/stvr

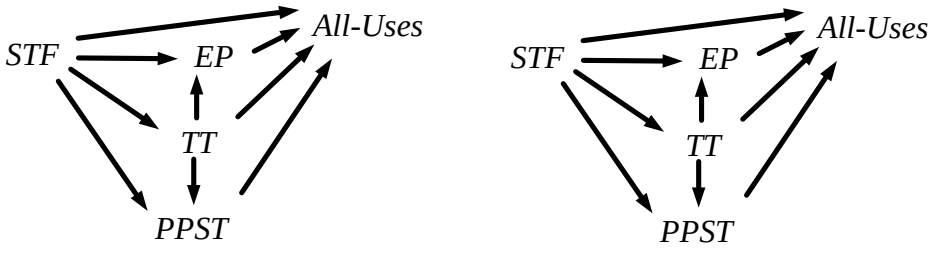
3.4.5. ***RQ5*** *results considering EFSM mutation scores MS and MS^*:* The correlation between MS and MS^* were analysed using both Person and Spearman correlation coefficients. The results of the correlation coefficients are 0.521 and 0.532 respectively. **These coefficients indicate somewhat weak association between MS and MS^* as, we recall that, strong correlation is indicated for coefficients above 0.7.**

3.4.6. ***RQ6*** *results considering code mutation scores MS and MS^* :* The correlation between MS and MS^* were analysed using both Person and Spearman correlation coefficients. The results of the correlation coefficients are 0.575 and 0.534 respectively. Both correlation coefficients indicate somewhat weak association between MS and MS^*.

3.5. Threats to validity

One threat to validity is to control the false discovery rate (controlling Type I error) that may occur, as a result of the considered multiple tests, in the statistical assessment. This threat was alleviated by the use of the Benjamini and Hochberg [8] method in which adjusted p-values are used instead of p-values during the assessment. Another threat is the adequacy of obtained results, and this was handled by using six real EFSM specifications when considering EFSM faults and twelve Java implementations of these specifications when considering code faults. Another threat is related to the derivation of test suites, mutants, and computation of corresponding mutation scores. This was addressed by the development and use of many related software tools as described in Section 3.2 and by the use of an appropriate mutations score that removes subsuming mutants before computing the score. **We note that we worked on deriving optimal (or near-optimal) length test suites, as mentioned in Section 3.2, that cover the intended test selection criteria. Yet, it is worth mentioning, as in [23, 24], that there is no correspondence between length (or number of test cases) of the EFSM test suites and their fault coverage (mutation scores). For instance; single-transfer fault and transitions tour tests outperformed EP and PPST though they are shorter (and have less number of tests), etc.** Finally, it is worth mentioning that though we are confident about the obtained results over specifications written as EFSMs with corresponding implementations written using the Java language; yet, unfortunately, we can not generalize these results to specifications written over other formalisms or to implementations using other programming languages or to general implementations that do not correspond to EFSMs. This is a natural limitation that emphasizes the need for conducting more related assessments.

4. SUMMARY OF OBTAINED RESULTS



(a) – Results over EFSM faults (**RQ1**) (b) – Results over code faults (**RQ2**)

An arrow, $TS_i \longrightarrow TS_j$ indicates that TS_i outperformed TS_j

Figure 4. The similar results obtained over (a) EFSM faults (**RQ1**) and (b) code faults (**RQ2**).

DOI: 10.1002/stvr

Figure 4 includes a summary of obtained results on assessing the considered test suites against EFSM (**RQ1**) and code faults (**RQ2**). The following outcomes and conclusions are obtained using these assessments and the other assessments considered in this paper.

Outcome 1: **For every pair of considered test suites; if a test suite outperformed another in covering EFSM faults then the test suite also outperformed the other in covering code faults. This result was obtained using the assessments of RQ1 and RQ2, respectively.**

Outcome 1 is very interesting as it allows us to uniquely determine, as given below, which EFSM test suites are better than others in detecting both EFSM and code faults. In addition, obtaining consistent results over the considered EFSM and code fault domains indicates that there is a strong correspondence between these domains. We think the reason behind this is that our Java implementations and their code mutants are special as they also have EFSM behavior. This result will open the door for more related work as will be mentioned in the Conclusion section.

Outcome 2: **Single transfer fault test suites (STF) outperformed all other test suites in covering both EFSM and code faults.**

Outcome 2 clearly shows that it is better to use a test suite that covers the single transfer faults of an EFSM than EP, PPST, and TT test suites which are based on some graph-based test selection criteria. This result is expected when considering the coverage of EFSM faults; as STF test suites are derived considering EFSM faults. However, obtaining the same results over code faults is interesting and it is due to the correspondence between EFSM and code faults as mentioned above for Outcome 1. So, a STF test suite derived taking into account EFSM faults captured more EFSM faults than other test suites derived based on covering the transitions of the EFSM. In addition, due to the correspondence, the STF test suite also captured more code faults than other test suites.

Outcome 3: **Transition tours outperformed both EP and PPST test suites.**

Outcome 3 shows that using a test suite that covers all transitions of a machine with a single test case (a transition tour) is better than using a test suite that covers these transitions using many test cases.

Outcome 4: Transition tours outperformed both EP and PPST test suites and no differences were spotted between edge-pair and prime path with side trip test suites in covering both EFSM and code faults. It is interesting to spot such result while considering test derivation from EFSM specifications.

Outcome 5: EFSM based all-uses test suites were outperformed by all other test suites in covering code faults.

This result is expected as the implementation can have many variables different than those of the specification. However, all-uses tests were also outperformed by all other tests when considering EFSM based faults. These results indicate that it is not beneficial to use data-flow testing when deriving tests from EFSM specifications. We understand this result as many transitions of an EFSM may not define nor use variables; i.e., defined only over non-parameterized input/output pairs.

Outcome 6: **For each pair of considered (EFSM/code) operators and their corresponding classes of (EFSM/code) faults, there was no significant difference of the mutation scores between these classes. These results were obtained by the assessments related to RQ3 and RQ4, respectively.**

Outcome 6 is interesting as it shows that the considered EFSM based test suites have no preferences in covering certain classes than others in covering EFSM or code faults. In fact, if this was not the case, then we would have to find which classes are usually less covered; and then argue that EFSM model-based testing is not appropriate for covering certain classes of faults.

DOI: 10.1002/stvr

Two other studies were conducted, using correlation analysis, over the considered test suites using traditional mutation scores and scores derived after removing subsuming mutants. According to these studies the following holds.

Outcome 7: There is a clear evidence that removing subsuming mutants has an impact on the mutation scores of the considered test suites. This result is validated considering EFSM mutants of the EFSM examples and the Java mutants of of the Java implementations of these specifications.

Thus, the conjecture provided in [52] is verified here for EFSM specifications and their corresponding EFSM and code faults. In addition, based on this results we decided, as suggested in [52], to remove subsuming mutants before computing the mutation scores in the provided assessments (**RQ1-RQ4**).

5. CONCLUSIONS AND FURTHER RESEARCH WORK

We provide novel empirical assessments of five different test suites derived from extended finite state machine (EFSM) specifications; namely, test suites that satisfy the all-uses, edge-pair, prime path with side trip, single transfer faults, and transition tour coverage criteria. The first assessment is done against EFSM mutants of six real EFSM examples while the second is carried out against the code mutants of twelve (Java) implementations of these examples. Two other assessments are provided to compare the coverage of the test suites over different classes of EFSM and code faults. A proper statistical evaluation method is used in the assessments. Namely, logit transformation of the percentage data (mutation scores) is used to expand the ends of the scale of the distribution and make it more symmetrical. Then testing the overall significance among the considered test suites is done. If an overall significance is detected, post hoc tests of pairwise comparisons are conducted, and the resulting p-values of the multiple tests are then adjusted, using the method in [8], to reduce the impact of Type-I error. In addition, two simple correlation studies are provided that show the impact of subsuming mutants on mutation scores; and accordingly, a proper mutation score that removes these mutants is considered in the assessments.

The outcomes of the assessments are summarized in a separate section. In nutshell, EFSM based all-uses tests were outperformed by all other test suites; and transition tours outperformed edge-pair and prime path with side trips; in addition, single transfer fault test suites are the best choice as they outperformed all other test suites.

A main outcome of the assessments is that consistent results are obtained over both EFSM and code fault domains. That is, if a test suite outperformed another in covering EFSM faults, then the test suite also outperformed the other in covering code faults. This result indicates that there is a strong correspondence between the considered EFSM and code fault domains. However, there is a need for more quantitative and analytical work to examine in detail this correspondence. Furthermore, according to the conducted assessments, test suites have similar coverage of different classes of EFSM and code faults. Thus, EFSM based tests are not biased against covering certain types of faults than others.

Though it is interesting to empirically assess the coverage of different EFSM test suites against code faults; as done in this paper, yet it would be also interesting to determine which faults are not detected by these test suites. Also, there is a need to validate the results obtained in this paper considering other programming languages; such as C and Python. Another research direction is to validate the results considering tests derived from other formal or semi-formal specification techniques, such as statecharts, extended labeled transition systems, etc, and their corresponding implementations.

Acknowledgements

The authors would like to thank Professor Tao Xie and the anonymous reviewers for their helpful comments that helped us improve the paper. Also, we thank the AUS undergraduate and graduate students who participated in deriving the Java implementations and in the development/use of the tools used in this work. We also thank Mr. Faiz Hasan for some related

DOI: 10.1002/stvr

tool development and for running the experiments. This work was partially supported by an AUS 2019 FRG.

REFERENCES

1. ABDURAZIK, A., AMMANN, P., DING, W., AND OFFUTT, A. J. Evaluation of three specification-based testing criteria. *Proceedings Sixth IEEE International Conference on Engineering of Complex Computer Systems. ICECCS 2000* (2000), 179–187.
2. ADITYA, M. *Foundation of Software Testing*, 2 ed. Addison-Wesley Professional, 2014.
3. AICHERNIG, B. K., BRANDL, H., JÖBSTL, E., KRENN, W., SCHLICK, R., AND TIRAN, S. Killing strategies for model-based mutation testing. *Software Testing, Verification and Reliability 25*, 8 (2015), 716–748.
4. AMMANN, P., DELAMARO, M. E., AND OFFUTT, J. Establishing theoretical minimal sets of mutants. In *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation* (2014), IEEE, pp. 21–30.
5. AMMANN, P., AND OFFUTT, J. *Introduction to Software Testing*, 1 ed. Cambridge University Press, New York, NY, USA, 2008.
6. ANDREWS, J. H., BRIAND, L. C., LABICHE, Y., AND NAMIN, A. S. Using mutation analysis for assessing and comparing testing coverage criteria. *IEEE Transactions on Software Engineering 32*, 8 (2006), 608–624.
7. BELLI, F., BUDNIK, C. J., HOLLMANN, A., TUGLULAR, T., AND WONG, W. E. Model-based mutation testing—approach and case studies. *Science of Computer Programming 120* (2016), 25 – 48.
8. BENJAMINI, Y., AND HOCHBERG, Y. Controlling the false discovery rate: a practical and powerful approach to multiple testing. *Journal of the Royal statistical society: series B (Methodological) 57*, 1 (1995), 289–300.
9. BOCHMANN, G., AND PETRENKO, A. Protocol testing: Review of methods and relevance for software testing. In *ACM International Symposium on Software Testing and Analysis* (Seattle USA, 1994), pp. 109–123.
10. BOURHFIR, C., ABDOULHAMID, E., KHENDEK, K., AND R, R. D. Test cases selection from SDL specifications. *Computer Networks 35(6)* (2001), 693–708.
11. BRIAND, L., DI PENTA, M., AND LABICHE, Y. Assessing and improving state-based class testing: A series of experiments. *Software Engineering, IEEE Transactions on 30* (12 2004), 770– 783.
12. BROWN, H., AND PRESCOT, R. *Applied mixed models in medicine*. Wiley Sons; Chichester, UK, 2nd ed. SAS Institute, Inc.; Cary, NC, 2006.
13. BUDD, T. A., AND ANGLUIN, D. Two notions of correctness and their relation to testing. *Acta Inf. 18*, 1 (Mar. 1982), 31–45.
14. CAVALLI, A., GERVY, C., AND PROKOPENKO, S. New approaches for passive testing using an extended finite state machine specification. *Inf. Softw. Technol. 45* (2003), 837–852.
15. CHEKAM, T. T., PAPADAKIS, M., LE TRAON, Y., AND HARMAN, M. An empirical study on mutation, statement and branch coverage fault revelation that avoids the unreliable clean program assumption. In *IEEE/ACM 39th ICSE* (May 2017), pp. 597–608.
16. CHEN, W. H. Executable test sequences for the protocol data flow property. In *Formal Techniques for Networked and Distributed Systems* (2011), pp. 285–299.
17. DEVROEY, X., PERROUIN, G., PAPADAKIS, M., LEGAY, A., SCHOBBENS, P.-Y., AND HEYMANS, P. Featured model-based mutation analysis. In *IEEE/ACM 38th ICSE* (New York, NY, USA, 2016), ACM, pp. 655–666.
18. DOROFEEVA, R., EL-FAKIH, K., MAAG, S., CAVALLI, A. R., AND YEVTUSHENKO, N. FSM-based conformance testing methods: a survey annotated with experimental evaluation. *Information and Software Technology 52*, 12 (2010), 1286–1297.
19. DUALE, A. Y., AND UYAR, M. U. A method enabling feasible conformance test sequence generation for EFSM models. *IEEE Transactions on Computers 53*, 5 (2004), 614–627.
20. EL-FAKIH, K., DOROFEEVA, R., YEVTUSHENKO, N., AND BOCHMANN, G. FSM-based testing from user defined faults adapted to incremental and mutation testing 1. *Programming and Computer Software 38*, 4 (2012), 201–209.
21. EL-FAKIH, K., KOLOMEEZ, A., PROKOPENKO, S., AND YEVTUSHENKO, N. Extended finite state machine based test derivation driven by user defined faults. In *Software Testing, Verification, and Validation, ICST* (2008), pp. 308–317.
22. EL-FAKIH, K., PROKOPENKO, S., YEVTUSHENKO, N., AND V. BOCHMANN, G. Fault diagnosis in extended finite state machines. In *Testing of Communicating Systems* (Berlin, Heidelberg, 2003), D. Hogrefe and A. Wiles, Eds., Springer Berlin Heidelberg, pp. 197–210.
23. EL-FAKIH, K., SALAMEH, T., AND YEVTUSHENKO, N. On code coverage of extended FSM based test suites: An initial assessment. In *26th ICTSS, Madrid, Spain, September 23-25* (2014), pp. 198–204.
24. EL-FAKIH, K., SIMÃO, A., JADOON, N., AND MALDONADO, J. C. An assessment of extended finite state machine test selection criteria. *Journal of Systems and Software 123* (2017), 106–118.
25. EL-FAKIH, K., YEVTUSHENKO, N., BOZGA, M., AND BENSALEM, S. Distinguishing extended finite state machine configurations using predicate abstraction. *J. Software Eng. R&D 4* (2016), 1.
26. FABBRI, S. C. P. F., MALDONADO, J. C., SUGETA, T., AND MASIERO, P. C. Mutation testing applied to validate specifications based on statecharts. In *Proceedings 10th International Symposium on Software Reliability Engineering (Cat. No.PR00443)* (1999), pp. 210–219.
27. FRANKL, P. G., AND DENG, Y. Comparison of delivered reliability of branch, data flow and operational testing: A case study. In *ISSTA* (2000).
28. FRANKL, P. G., AND IAKOUNENKO, O. Further empirical studies of test effectiveness. In *Proceedings of the 6th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (New York, NY, USA, 1998), SIGSOFT'98/FSE-6, ACM, pp. 153–162.

DOI: 10.1002/stvr

29. FRANKL, P. G., AND WEISS, S. N. An experimental comparison of the effectiveness of branch testing and data flow testing. *IEEE Transactions on Software Engineering 19*, 8 (Aug 1993), 774–787.
30. FRANKL, P. G., WEISS, S. N., AND HU, C. All-uses vs mutation testing: An experimental comparison of effectiveness. *Journal of Systems and Software 38*, 3 (1997), 235 – 253.
31. FRANKL, P. G., AND WEYUKER, E. J. An applicable family of data flow testing criteria. *IEEE Transactions on Software Engineering 14*, 10 (1988), 1483–1498.
32. FRIEDMAN, M. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of American Stistical Association 32* (1937), 675–70.
33. HENARD, C., PAPADAKIS, M., PERROUIN, G., KLEIN, J., AND TRAON, Y. L. Assessing software product line testing via model-based mutation: An application to similarity testing. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops* (2013), pp. 188–197.
34. HOGREFE, D. Osi formal specification case study: the inres protocol and service. In *IAM-91-012 - Technical Report: University of Bern, Switzerland* (1992).
35. HUTCHINS, M., FOSTER, H., GORADIA, T., AND OSTRAND, T. Experiments on the effectiveness of dataflow- and control-flow-based test adequacy criteria. In *Proceedings of 16th International Conference on Software Engineering* (1994), pp. 191–200.
36. JIA, Y., AND HARMAN, M. An analysis and survey of the development of mutation testing. *IEEE Trans. Softw. Eng. 37*, 5 (Sept. 2011), 649–678.
37. JUST, R. The major mutation framework: Efficient and scalable mutation analysis for java. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis* (New York, NY, USA, 2014), ISSTA 2014, ACM, pp. 433–436.
38. JUST, R., JALALI, D., AND ERNST, M. D. Defects4j: A database of existing faults to enable controlled testing studies for java programs. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis* (New York, NY, USA, 2014), ISSTA 2014, Association for Computing Machinery, p. 437–440.
39. JUST, R., JALALI, D., INOZEMTSEVA, L., ERNST, M. D., HOLMES, R., AND FRASER, G. Are mutants a valid substitute for real faults in software testing? In *22nd ACM SIGSOFT FSE* (2014), pp. 654–665.
40. KALAJI, A. S., HIERONS, R., AND SWIFT, S. An integrated search-based approach for automatic testing from extended finite state machine (EFSM) models. *Inf. Softw. Technol. 53* (2011), 1297–1318.
41. KALAJI, A. S., HIERONS, R. M., AND SWIFT, S. Generating feasible transition paths for testing from an extended finite state machine (EFSM) with the counter problem. In *2010 Third International Conference on Software Testing, Verification, and Validation Workshops* (2010), pp. 232–235.
42. KANSOMKEAT, S., OFFUTT, J., ABDURAZIK, A., AND BALDINI, A. A comparative evaluation of tests generated from different uml diagrams. In *2008 Ninth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing* (2008), pp. 867–872.
43. KINTIS, M., PAPADAKIS, M., PAPADOPOULOS, A., VALVIS, E., MALEVRIS, N., AND LE TRAON, Y. How effective are mutation testing tools? an empirical analysis of java mutation testing tools with manual analysis and real faults. *Empirical Software Eng. 23*, 4 (2018), 2426–2463.
44. KOVÁCS, G., PAP, Z., LE VIET, D., WU-HEN-CHANG, A., AND CSOPAKI, G. Applying mutation analysis to SDL specifications. In *SDL 2003: System Design* (Berlin, Heidelberg, 2003), R. Reed and J. Reed, Eds., Springer Berlin Heidelberg, pp. 269–284.
45. KRENN, W., SCHLICK, R., TIRAN, S., AICHERNIG, B., JÖBSTL, E., AND BRANDL, H. MoMut::UML model-based mutation testing for UML. *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)* (2015), 1–8.
46. LI, N., AND OFFUTT, J. An empirical analysis of test oracle strategies for model-based testing. In *2014 IEEE Seventh International Conference on Software Testing, Verification and Validation* (March 2014), pp. 363–372.
47. LI, N., PRAPHAMONTRIPONG, U., AND OFFUTT, J. An experimental comparison of four unit test criteria: Mutation, edge-pair, all-uses and prime path coverage. In *2009 International Conference on Software Testing, Verification, and Validation Workshops* (2009), pp. 220–229.
48. LI, N., PRAPHAMONTRIPONG, U., AND OFFUTT, J. An experimental comparison of four unit test criteria: Mutation, edge-pair, all-uses and prime path coverage. In *2009 International Conference on Software Testing, Verification, and Validation Workshops* (2009), IEEE, pp. 220–229.
49. MUKAKA, M. Statistics corner: a guide to appropriate use of correlation coefficient in medical research. *Malawi Med J. 24* (2012), 69–71.
50. OFFUTT, A. J., PAN, J., TEWARY, K., AND ZHANG, T. An experimental evaluation of data flow and mutation testing. *Software: Practice and Experience 26*, 2 (1996), 165–176.
51. OVERHOLSER, B., AND SOWINSKI, K. Biostatistics primer: Part 2. *Nutrition in Clinical Practice 23* (2008), 76–84.
52. PAPADAKIS, M., HENARD, C., HARMAN, M., JIA, Y., AND LE TRAON, Y. Threats to the validity of mutation-based test assessment. In *Proceedings of the 25th International Symposium on Software Testing and Analysis* (New York, NY, USA, 2016), ISSTA 2016, ACM, pp. 354–365.
53. PETRENKO, A., BORODAY, S., AND GROZ, R. Confirming configurations in EFSM testing. *IEEE Transactions on Software Engineering 30*, 1 (2004), 29–42.
54. PINTO FERRAZ FABBRI, S. C., DELAMARO, M. E., MALDONADO, J. C., AND MASIERO, P. C. Mutation analysis testing for finite state machines. In *Proceedings of 1994 IEEE International Symposium on Software Reliability Engineering* (1994), pp. 220–229.
55. RAMALINGOM, T., THULASIRAMAN, K., AND DAS, A. Context independent unique state identification sequences for testing communication protocols modelled as extended finite state machines. *Computer Communications 26* (2003), 1622–1633.
56. RFC1350. The tftp protocol (revision 2). In *URL: http://www.rfc-editor.org/rfc/rfc1350.txt*.
57. RFC1939. Post office protocol - version 3,.
58. SEIFERT, D. Conformance testing based on uml state machines. In *Formal Methods and Software Engineering*

DOI: 10.1002/stvr

(2008), pp. 44–65.
59. SHAFFER, J. Multiple hypothesis testing. *Annual Review of Psychology 46* (1995), 561–584.
60. SHAPIRO, S., AND WILK, M. An analysis of variance test for normality (complete samples). *Biometrika 52* (1965), 591–611.
61. SIMÃO, A., AMBRÓSIO, A., FABBRI, S., DO AMARAL, A., MARTINS, E., AND MALDONADO, J. Plavis/FSM: an environment to integrate FSM-based testing tools. In *Tool Session of XIX Brazilian Symposium on Software Engineering* (2008), pp. 1–6.
62. SUGETA, T., MALDONADO, J. C., AND WONG, E. Mutation testing applied to validate SDL specifications. In *Testing of Communicating Systems* (2004), pp. 193–208.
63. TURLEA, A., IPATE, F., AND LEFTICARU, R. Generating complex paths for testing from an EFSM. *2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)* (2018), 242–249.
64. URAL, H., AND WILLIAMS, A. Test generation by exposing control and data dependencies within system specifications in SDL. In *Formal Description Techniques* (1993), pp. 335–350.

DOI: 10.1002/stvr