IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 25 October 2022, accepted 25 December 2022, date of publication 4 January 2023, date of current version 9 January 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3234186

RESEARCH ARTICLE

Incremental Testing in Software Product Lines–An Event Based Approach

MUTLU BEYAZIT[1], TUGKAN TUGLULAR[2], (Member, IEEE), AND DILEK ÖZTÜRK KAYA[2]
[1]Department of Computer Engineering, Faculty of Engineering, Yaşar University, 35100 İzmir, Türkiye
[2]Department of Computer Engineering, Faculty of Engineering, İzmir Institute of Technology, 35430 İzmir, Türkiye

Corresponding author: Tugkan Tuglular (tugkantuglular@iyte.edu.tr)

This work was supported by the Scientific and Technological Research Council of Türkiye (TÜBİTAK) under Grant 117E884.

ABSTRACT One way of developing fast, effective, and high-quality software products is to reuse previously developed software components and products. In the case of a product family, the software product line (SPL) approach can make reuse more effective. The goal of SPLs is faster development of low-cost and high-quality software products. This paper proposes an incremental model-based approach to test products in SPLs. The proposed approach utilizes event-based behavioral models of the SPL features. It reuses existing event-based feature models and event-based product models along with their test cases to generate test cases for each new product developed by adding a new feature to an existing product. Newly introduced featured event sequence graphs (FESGs) are used for behavioral feature and product modeling; thus, generated test cases are event sequences. The paper presents evaluations with three software product lines to validate the approach and analyze its characteristics by comparing it to the state-of-the-art ESG-based testing approach. Results show that the proposed incremental testing approach highly reuses the existing test sets as intended. Also, it is superior to the state-of-the-art approach in terms of fault detection effectiveness and test generation effort but inferior in terms of test set size and test execution effort.

INDEX TERMS Incremental testing, model-based testing, software product line.

I. INTRODUCTION

A set of products with common features with varying additional features related to each other in a specific domain constitutes a software product line (SPL) [1]. The software product line paradigm enables systematic reuse of software assets resulting in faster development and increased product quality [2]. SPLs are welcome to constant changes and improvements in design and development. Therefore, the methods ensuring their quality should be able to go along with the same paradigm.

In SPLs, there are different products of software equipped with different features to appeal to different target audiences; for example, standard, professional and enterprise versions of a software product. Every new feature added to the features at the base/core of the software increases the complexity of the software and thereby reinforces its predisposition to failure [3]. The features in a software product line shape the products according to the selection of the stakeholders and enable us to distinguish one product from another [4]. The feature selection can be performed using feature diagrams [5].

In software product lines, software testing comprises testing components in the asset pool (i.e., domain tests) and testing the final product (i.e., application tests). We suggest using model-based test generation methods for the automatic generation of domain tests because domain engineering mainly utilizes model-based approaches. In software product lines, applications are built by reusing existing components and their configurations. Similarly, application tests should be generated by reusing domain tests and their configurations.

In this paper, we propose a model-based approach to functional or black-box testing of SPLs incrementally. The idea is based on using the previously generated test artifacts (models, tests, etc.) of existing products to generate the test artifacts for the new products each of which includes an additional feature. In order to realize this idea, it is possible to use models of different focus and expressive power. We select an event-based modeling approach using event sequence graphs (ESGs) [6].

The associate editor coordinating the review of this manuscript and approving it for publication was Yang Liu.

ESGs are equivalent to finite state machines (FSMs) [7], one of the most commonly used state-based models in testing practice. An FSM can be converted to an ESG, and vice versa, making ESGs as practical as FSMs. The difference, however, is that ESGs focus on events and, therefore, have simpler semantics as well as simpler associated fault models [8]. There is a considerably large body of work done on ESGs [6], [9], [10] and similar event-based models [8], [11], [12], [13].

This work introduces an event-based incremental testing approach for testing products of SPLs. One of the main novelties of the approach is that featured event sequence graphs (FESGs) which are extended from ESGs, are used to model a product based on its features, and explicitly capture behavioral variability in SPLs. More precisely, the event-based behavior of each feature is modeled as a kind of ESG, called feature ESG (f-ESG), and the behavior of the product is defined by combining f-ESGs of its features. Another novelty is that FESG and test sequences of an existing product are reused in test generation together with f-ESG of a new feature that is to be added to the existing product to make up a new product. First, test sequences for the new feature are generated by using the f-ESG of the new feature and the FESG of the existing product. Then, test sequences of the existing product are composed with those of the new feature incrementally to obtain test sequences for the new product.

Note that we introduced FESGs in our previous work [14]. We also presented a test generation algorithm with limitations to generate tests for a product by composing all the sequences generated from its f-ESGs. In this paper, we extend [14] in two directions. One is redesigning the approach to work with increments to promote reuse. Although our previous work is called ''incremental'', it is ''compositionally incrementa'', where f-ESGs of the features of a new product are processed separately. The approach presented in this paper reuses the FESG of an existing product together with the f-ESG of an additional feature that makes up a new product. The other direction for extension [14] is proposing a more efficient and general test generation algorithm. The previous algorithm generates test cases from the f-ESGs of a new product separately and then composes them to obtain test cases for the new product. The proposed algorithm, however, generates test cases for the new product by making use of the FESG of an existing product and the f-ESG of an additional feature. This algorithm also handles the cases that are not handled by the previous one where sequences cannot be composed in the middle parts and all sequences generated from the ESG model of core feature end with the same finish event.

The advantages of the proposed approach are twofold. First, with separate behavioral modeling of features, when a change occurs in a feature or when a product with a new feature emerges, the tester works only on the affected feature ESG instead of the affected product ESGs, each of which could be considerably larger than a feature ESG, or only on the new feature ESG instead of the new and larger product ESG. Second, by applying an incremental test generation process, the tester can generate tests by composing the sequences of the affected feature or the sequences of the new feature incrementally with existing sequences. Thus, the proposed approach becomes more adept at handling constant changes and the emergence of new products when compared to the state-of-the-art ESG-based approach [6], which uses a single ESG (of the product) to generate tests, and to the approach in our previous work [14], which composes sequences of all feature ESGs (of the product) to generate tests.

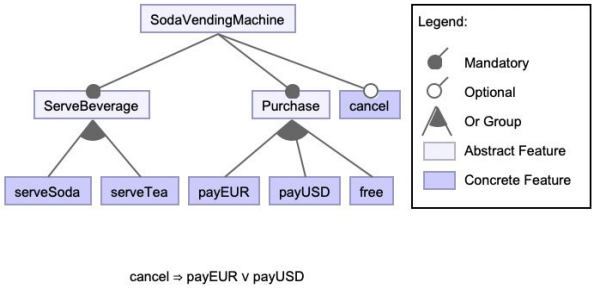


FIGURE 1. Soda Vending Machine SPL feature diagram (modified from [15]).

The paper is organized as follows. Section II introduces the foundations of feature modeling and the state-of-the-art test generation approach using ESGs. The proposed approach, incremental test generation using FESGs, is described in Section III and Section IV validates the approach by performing evaluations to analyze its characteristics in comparison to the state-of-the-art ESG-based testing approach. Related work is discussed in in Section V and Section VI concludes the paper and outlines the future work.

II. PRELIMINARIES

This section introduces our running SPL example which is used paper and presents a brief background on event-based modeling and test generation using event sequence graphs (ESGs).

A. RUNNING EXAMPLE: SODA VENDING MACHINE SPL

Feature diagrams define the configuration options and the dependencies of features in SPLs [5]. The root represents the core of the SPL and the nodes represent features, which can be either mandatory or optional, and are denoted by filled and empty small circles, respectively. The features are classified as concrete and abstract. Concrete features correspond to real features in an SPL whereas abstract features are used to group features. Two or more features with an OR relationship can exist in a product in different combinations, whereas an XOR relationship allows only one of them to exist in a product. XOR relationship models alternative features.There are also require and exclude relationships, where the former denotes a feature that cannot exist without the required feature and the latter denotes a feature that omits excluded feature.

In the feature diagram given in Fig. 1, Soda Vending Machine SPL is shown with mandatory abstract features

Algorithm 1: Sequence Generation

Input: G = (V, E, Ξ, Γ) – an ESG
Output: T – a set of complete sequences for G
1 G_{SC} = add an edge from ']' to '[' into G
2 G_{SC-B} = add necessary paths until degree of each vertex in G_{T-SC} is 0
3 T = compute Euler cycles from G_{T-SC-B}

which are *ServeBeverage* and *Purchase*. In this diagram, the abstract feature *ServeBeverage* is grouping the features *serveSoda* and *serveTea* with OR relationship similar to *Purchase* which groups the features *payEUR*, *payUSD*, and *free*. The feature *cancel* is optional for this SPL. Furthermore, the implication that is written below the feature diagram corresponds to require relationship where the feature *cancel* requires feature *payEUR* or feature *payUSD* in a product configuration. Soda Vending Machine SPL example is modified from [15] and all of the f-ESG models are given in external resources.[1]

B. TEST GENERATION USING ESGs

An ESG is used for behavioral modeling of systems [16]. ESGs focus on the externally observable behavior of computer-based systems by means of discrete event-based models [17]. Definition 1 gives a definition of ESGs.

Definition 1: An event sequence graph (ESG) is a tuple (V, E, Ξ, Γ), where V≠∅ is a finite set of nodes (vertices or events) and E⊆V×V is a finite set of arcs (edges), and Ξ, Γ⊆V finite sets of distinguished nodes with $\xi \in \Xi$ called entry nodes (start events) and $\gamma \in \Gamma$ called exit nodes (finish events) [9].

The entry and exit vertices of an ESG are marked using pseudo vertices. Each start vertex $\xi \in \Xi$ is preceded by pseudo start vertex '[' and each finish vertex $\gamma \in \Gamma$ is followed by pseudo finish vertex ']' [10]. Pseudo vertices and edges related to pseudo vertices (pseudo edges) are included in V and E, respectively. An example ESG is given in Fig. 2.

Definition 2: Let (V, E, Ξ, Γ) be an ESG as defined in Definition 1. Any sequence of nonpseudo vertices $v_0, v_1, \ldots, v_k$ is called an event sequence (ES) if $(v_i, v_{i+1}) \in$E for i=0, …, k-1 [6], [9], [10]. An ES of length k is called k-sequence [13]. An ES $v_0, v_1, \ldots, v_k$ is a start sequence if $v_0 \in \Xi$ is an entry and it is a finish sequence if $v_k \in \Gamma$ is an exit [8]. An ES is a complete event sequence (CES) if it is both a start and a finish sequence [18].

The state-of-the-art test case generation from an ESG is based on solving the Chinese postman problem (CPP) [19], [20]. Solving CPP requires finding the shortest circuit that visits every edge in a given graph. The state-of-the-art implementation[2] to solve CPP on ESGs computes Euler cycles [21] in the graph using Hierholzer Algorithm [22]. Euler cycles are cycles starting from and returning back to the same vertex by visiting each edge exactly once. Each Euler cycle corresponds to a CES and represents a test case. The resulting test set is optimal in the sense that each edge is covered a minimum number of times. Algorithm 1 is adapted from [6]; it outlines the generation of test cases from a given ESG in an optimal manner.

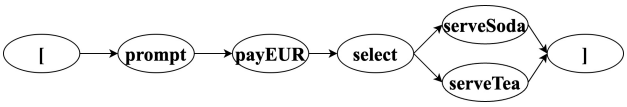


FIGURE 2. *EUR product* ESG of soda vending machine SPL.

Algorithm 1 firstly, constructs a strongly connected [19] ESG by adding an edge from ']' to '[' to the input ESG and, then, builds a balanced, i.e. symmetric [19] ESG from the strongly connected ESG by adding paths until the degree of each vertex is 0. In the last step, Algorithm 1 generates Euler cycles [19] from the strongly connected and balanced ESG. These generated Euler cycles are the sequences of the input ESG. The worst-case time complexity of Algorithm 1 is $O(|V|^3)$ [6] where V is the vertex set of the input ESG. For more discussion on the ESG-based test generation algorithms, reader may refer to [6], [9], [10], and [17].

III. INCREMENTAL TEST GENERATION USING FEATURED ESGs

In this section, we discuss our proposed incremental test generation approach. It exploits the fact that test cases for certain products already exist and they can be reused to generate test cases for new products. New products are related to the existing products in such a way that each new product can be obtained from an existing product by adding a new feature.

Featured ESGs (FESGs) are employed to propose an incremental test generation approach. Based on the definition of an ESG (see Definition 1), definition of a FESG is given in Definition 3.

Definition 3: A featured event sequence graph (FESG) is (F, c, Ξ, Γ) where F=$\{f_1, f_2, \ldots, f_N\} \neq \emptyset$ is a finite set of ESGs called feature ESGs (f-ESGs) with each f_i=(V_i, E_i, Ξ_i, Γ_i). c∈F is a special f-ESG called core ESG (c-ESG). Ξ,Γ⊆$\cup_{i=1}^{N} V_i$ are finite sets of distinguished nodes called entry nodes (start events) and exit nodes (finish events), respectively.

Definition 3 suggests that a FESG contains a set of feature ESGs (f-ESGs) one of which is designated as the core ESG (c-ESG). A feature diagram (as given in Fig. 1) is used to learn the relation of each feature to the other features during the creation of f-ESGs of the features. f-ESGs of features that together form a particular product are added into a FESG to create a behavioral model for the product.

An f-ESG of a given FESG, except for the c-ESG, contains one or more vertices of other f-ESGs, called connection

[1] https://github.com/esg4aspl/SPL-FESG-Examples/blob/master/SodaVendingMachineSPL.md [accessed 10 August 2022]
[2] TSD(2022). Download TestSuiteDesigner [online]. Website http://download.ivknet.de/ [accessed 10 August 2022]

Algorithm 2: Incremental Sequence Generation

Input: G = (F, c, Ξ, Γ) – an FESG T – a set of complete sequences for G f_{new} = (V_{new}, E_{new}, Ξ_{new} Γ_{new}) - a new f-ESG to be added to G

Output: T' – a set of complete sequences for the new FESG G'

1 G' = (F∪{f_{new}}, c, Ξ∪Ξ_{new}, Γ∪Γ_{new})
2 T_{new} = generateSequences(f_{new}) // See Algorithm 1
3 T' = composeSequences(T, T_{new}, Ξ ∪ Ξ_{new}, Γ ∪ Γ_{new}) // See Algorithm 3

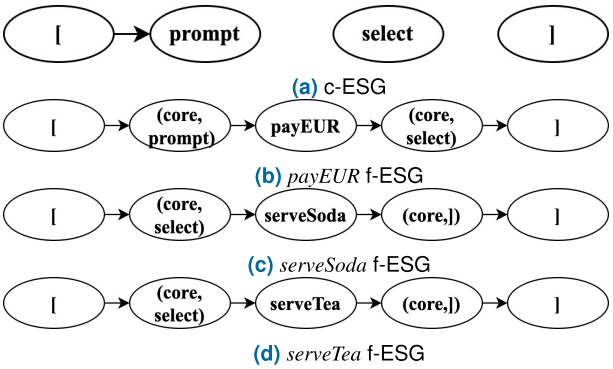


FIGURE 3. F-ESGs of *EUR product* of soda vending machine SPL.

events [14], to express that it requires events of these other f-ESGs. c-ESG of the FESG includes events that are (directly or indirectly) required by f-ESGs of the given FESG; it does not contain any connection events. Connection events can be shown as (f-ESG Name, Event Name) pairs. f-ESG names and event names are used as f-ESG identifiers and event identifiers, respectively. Connection events are used considering the constraints that are defined in the feature diagram (as given in Fig. 1).

When compared to an ESG, an f-ESG does not need to satisfy the following conditions which have to be satisfied by an ESG: (1) each event is reachable from the pseudo start vertex, and (2) the pseudo finish vertex is reachable from each event. Also, in terms of expressive power, f-ESGs are semantically richer than ESGs because an f-ESG possibly contains events of another f-ESG (called connection events) and, thus, express the dependency relation between these f-ESGs. This increased expressiveness, however, does not make the test generation harder because sequences of each f-ESG are generated by applying an ESG-based test generation and the information on connection events are used in the sequence composition phase (see Algorithm 2). In addition, for the construction costs, one can say that constructing f-ESGs of a given product has almost the same complexity as constructing the product ESG because the same behavior is modeled and, in total, almost the same number and type of model elements are used.

Example 1: EUR product ESG is given in Fig. 2. Since EUR product includes *payEUR*, *serveSoda* and, *serveTea* features, its FESG (F, c, Ξ, Γ) is defined as follows.

- F = {c, f_1, f_2, f_3}
  - c is c-ESG (see Fig. 3a) such that V_c={prompt, select}, E_c={ }, Ξ_c= {prompt} and Γ_c={ }
  - f_1 is payEUR f-ESG (see Fig. 3b) such that V_1 = {(core, prompt), payEUR, (core, select)}, E_1 = {((core, prompt), payEUR), (payEUR, (core, select))}, Ξ_1 = { }, Γ_1 = { }
  - f_2 is serveSoda f-ESG (see Fig. 3c) such that V_2 = {(core, select), serveSoda, (core,])}, E_2 = {((core, select), serveSoda), (serveSoda, (core,]))}, Ξ_2 = { } and Γ_2 = {serveSoda}
  - f_3 is serveTea f-ESG (see Fig. 3d) such that V_3 = {(core, select), serveTea, (core,])}, E_3 = {((core, select), serveTea), (serveTea, (core,]))}, Ξ_3 = { } and Γ_3 = {serveTea}
- Ξ = $\Xi_1 \cup \Xi_2 \cup \Xi_3 \cup \Xi_c$ = {prompt}
- Γ = $\Gamma_1 \cup \Gamma_2 \cup \Gamma_3 \cup \Gamma_c$ = {serveSoda, serveTea}
  serveSoda and *serveTea* f-ESGs have connections to core f-ESG using (core, select) and (core,]) connection events.

The purpose of incremental test generation is to reuse FESG and test cases of an existing product to generate test cases for a new product which is obtained by including a new feature in the existing product. To achieve this, ESG-based test generation is applied to the f-ESG of the new feature, and its sequences are obtained. Later, these sequences are composed with the sequences of the existing product to obtain sequences for the new product. Algorithm 2 shows the relevant steps.

Algorithm 2 constructs a new FESG by joining the existing FESG's features, the entry nodes and the exit nodes with the new feature set and the new features' entry and exit nodes. Then, it generates test sequences from a new f-ESG using Algorithm 1 and composes these sequences with existing ones using Algorithm 3.

In order to show how Algorithm 2 works, we use *EUR product* as the existing product with a FESG model and a test set. *cancel* feature is added to *EUR product* to obtain *EUR-Cancel product* and Algorithm 2 is used to generate test cases for *EUR-Cancel product* in two steps: Example 2 for Algorithm 1 and Example 3 for Algorithm 3.

Example 2: When Algorithm 1 is applied to the f-ESG of *cancel* given in Fig. 4, the sequences "(payUSD,payUSD), cancel, returnMone" and "(payEUR,payEUR), cancel, returnMoney" are obtained.

After partial sequences are generated from a new f-ESG, Algorithm 3 can be used to compose them with existing complete sequences which are generated for an existing FESG model.

Algorithm 3 uses four different procedures to perform sequence composition: "initializeStartSequences", "updateStartSequences", "initializeCompleteSequences"

Algorithm 3: Sequence Composition

Input: T - a set of existing complete sequences T_{new} - a set of new partial sequences Ξ - a set of start vertices Γ - a set of finish vertices

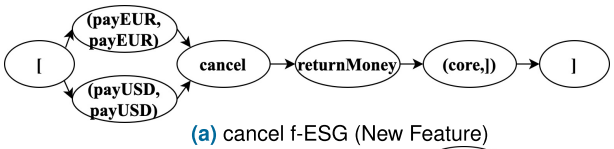
Output: CS – a set of composed complete sequences

```
SS = initializeStartSequences(T, T_new, Ξ)
notfinished = true
while notfinished is true do
    notfinished = updateStartSequences(T_new, SS)
CS = initializeCompleteSequences(SS, Γ)
notfinished = true
while notfinished is true do
    notfinished = updateCompleteSequences(SS, CS)
```

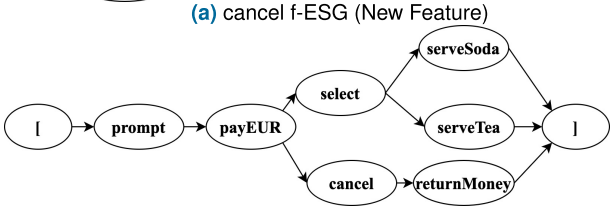
and "updateCompleteSequences". For the sake of saving space, we do not give separate algorithms for these procedures but explain them. "initializeStartSequences" constructs a set of start sequences SS. For each sequence s∈T∪T_{new}, if s is a start sequence, it is removed from the set that contains it. s is added in SS if it increases the coverage. "updateStartSequences" goes through the remaining sequences in T_{new}. For each s∈T_{new}, it tries to find a start sequence seq∈SS such that s can be completed to a start sequence seq_{new} by using a prefix of seq. If this is possible, s is removed from T_{new} and, if seq_{new} increases coverage, it is included in SS. Furthermore, if seq is a prefix of seq_{new}, seq is removed from SS. "updateStartSequences" is repetitively called until no more sequences remain in T_{new}. "initializeCompleteSequences" constructs a set of complete sequences CS. For each sequence s ∈ SS, if s is also a finish sequence, it is removed from SS and included in CS. "updateCompleteSequences" goes through the remaining start sequences in SS. For each s∈SS, it tries to find a complete sequence seq∈CS such that s can be completed to a finish sequence seq_{new} by using a suffix of seq. If this is possible, s is removed from SS and seq_{new} is included in CS. "updateCompleteSequences" is repetitively called until no more sequences remain in SS.

Example 3: Let T = {B1, B2} be a set of existing complete sequences of *EUR product* where B1 = "prompt, payEUR, select, serveSoda" and B2 = "prompt, payEUR, select, serveTea". Let T_{new} be the set of partial sequences of cancel f-ESG given in Example 2, and let Ξ = {prompt} be the start event set and Γ = {serveSoda, serveTea} be the finish event set. When Algorithm 3 is executed, the resulting set of complete sequences is CS = {C1, C2, C3} where C1 = "prompt, payEUR, select, serveSoda", C2 = "prompt, payEUR, select, serveTea" and, C3 = "prompt, payEUR, cancel, returnMoney" and. These sequences belong to EUR-Cancel product whose ESG is given in Fig. 4b. This product is obtained by adding cancel feature to *EUR product*.

In order to analyze the worst-case runtime complexity of Algorithm 2, we start by analyzing that of Algorithm 3.



(a) cancel f-ESG (New Feature)



(b) EUR-Cancel product ESG (New Product)

FIGURE 4. F-ESG of a new feature and ESG of a new product.

However, in order to better relate the complexities of Algorithm 2 and Algorithm 3 to that of Algorithm 1, we have the following for our incremental testing scenarios.

- The size of T_{new} is very small and bounded by a constant.
- The number of start events and the number of finish events are comparatively small and bounded by constants. (Even if not, they have very negligible effects.)
- $|T| = O(len(T)) = O(|E|)$ where $|T|$ is the number of sequences in T, $len(T)$ is the sum of the length of all sequences in T, and $|E|$ is the number of edges in the ESG which is used to generate T. Note that there is no work that relates the size of the input ESG to the size of the output test set. We assume that $len(T) <= c|E|$ (c is constant) because ESG-based test generation aims to cover the edges of the input ESG in an optimized manner.
- All sequences in T are start sequences because they are existing sequences.

Thus, in the worst-case scenario for Algorithm 3:

- "initializeStartSequences" is $O(|E|)$, because one needs to go through all the events of each sequence in $T \cup T_{new}$ to check the coverage contribution of the sequence.
- Before the loop on "updateStartSequences", T_{new} should be as large as possible (which implies that $SS = T$) and, at each iteration of the loop, the last sequence in T_{new} should be removed to construct a new start sequence by using the last sequence in SS. Thus, the loop on "updateStartSequences" runs at worst in $O(|E|)$ time.
- "initializeCompleteSequences" runs in $O(|T|) = O(len(T)) = O(|E|)$ time because one goes through all sequence in SS with $|SS| = O(|T|)$ and picks the starting sequences.
- Before the loop on "updateCompleteSequences", SS should be as large as possible (which implies that $|CS| = 1$) and, at each iteration of the loop, the last sequence in SS should be removed to form a new complete sequence using the last sequence in CS. Thus, the loop on "updateCompleteSequences" runs at worst in $O(|T||E|) = O(|E|^2)$ time.

Consequently, the worst-case runtime complexity of Algorithm 2 is $O(|T_{new}|^3 + |E|2) = O(|E|2)$. Note that a better upper bound can be obtained and the effect of the hidden constant can be discovered by analyzing the relations between the sizes of ESGs, f-ESGs, ESG test sets, and, f-ESG test sets experimentally, which is out of the scope of this paper.

IV. EVALUATIONS

In this section, evaluations are performed using three different SPLs and uniformly sampling 90 testing scenarios at random to validate the proposed incremental testing approach (Section III), to analyze its characteristics, and to compare it against the single-model-based approach (Section II-B).

A. APPROACHES UNDER CONSIDERATION

Testing scenarios which the incremental testing approach proposed in this paper is designed for can be characterized as triples (E, f, P). Here, E is an existing product (EP), f is a new feature to be added to E, and P is a product under consideration (PUC) which is obtained by adding f to E. Note that test models and the test set of E exist, and the test set of P is to be generated by reusing the test set of E and the test set of f. Such testing scenarios are relevant in the sense that they represent typical situations where a new product emerges in an SPL, and it is to be tested functionally using existing test artifacts.

For each testing scenario, testing approaches compared in the evaluations are defined as *inc* and *sm*. *inc* generates test sequences by composing test sequences of a new feature with existing test sequences of an EP as described in Section III (''incremental'' approach). *sm* generates test sequences in an optimized manner by using a single model as described in Section II-B (''single-model'' approach) where the model is obtained by composing all the related feature ESGs.

Note that, in the evaluations, our main purpose is to compare our proposed method which exploits the incremental nature of SPLs (inc) against a method which does not do that (sm). We exclude the method we proposed in our previous work [14], called *pm*, from the evaluations because it does not exploit the incremental nature of SPLs. More precisely, *pm* uses partial models, that is, feature ESGs, of a product to generate sequences first and then compose these sequences to obtain sequences for the product. Thus, it can actually be considered as ''partial-models-based'' test generation method. Furthermore, *pm* has two limitations which prevent us from using it in our evaluations: (1) sequences cannot be composed in the middle parts, and (2) all sequences generated from the core feature ESG are assumed to end with the same finish event.

B. RESEARCH METHODOLOGY

Our goals are to discern at which level *inc* reuses existing test sets, to see how different *inc* test sets are from the optimal *sm* test sets in terms of size, and to compare *inc* against *sm* in terms of fault detection and testing costs. To realize these goals, we ask the following research questions.

RQ1: What is reuse level of existing products' test sets by *inc*, and what is size difference of *inc* test sets from *sm* test sets?

RQ2: How does fault detection effectiveness of *inc* compare against that of *sm*?

RQ3: How does testing cost effectiveness of *inc* compare against that of *sm*?

Note that *sm* generates test sets in an optimized manner. Therefore, we expect that *inc* results in larger test sets and greater test execution costs. However, we do not know to which level the differences are and whether there is a good compromise with respect to the fault detection effectiveness. Also, although many use test set size as an indicator of test execution costs, our experience shows that it is not always true [7], [23]. Thus, we perform realistic test executions to obtain a better insight into the testing costs.

TABLE 1. Data on SPLs.

SPL	Number of Features	Number of Possible Products	Number of Possible Testing Scnearios
BA	9	42	97
eM	5	24	52
SAS	20	2664	10236

C. SPLs UNDER CONSIDERATION AND TESTING SCENARIOS

We use three SPLs: two from SPL2go and one from SPLOT (SPL2go and SPLOT are publicly available repositories of SPLs to be used for product-line analyses). We pick the SPLs in such a way that they have different numbers of features, different numbers of possible products, and different numbers of testing scenarios: Bank Account (BA) SPL,[3] e-Mail (eM) SPL[4] and Student Attendance System (SAS).[5] The SPLs BA and eM in SPL2go are also used in other works on SPL research [24], [25], [26]. All the features of the chosen SPLs are modeled using f-ESGs. Table 1 gives some data on these SPLs.

In the evaluations, we select 30 testing scenarios uniformly at random for each of the three SPLs so that we can use parametric statistical hypothesis tests to determine whether or not there is a significant difference between *inc* and *sm* approaches [27], [28]. Each testing scenario has an EP and a PUC; PUCs are used to collect data on the approaches. Due to the lack of space, data on on the models of the

[3]SPL2GO(2022). Catalog of SPLs [online]. Website http://spl2go.cs.ovgu.de/projects/54 [accessed 10 August 2022]

[4]SPL2GO(2022). Catalog of SPLs [online]. Website http://spl2go.cs.ovgu.de/projects/17 [accessed 10 August 2022]

[5]SPLOT(2022). SPL Online Tools [online]. Website http://www.splot-research.org/ [accessed 10 August 2022]

TABLE 2. Exemplary data on test generation and execution.

SPL	PUC ID	Approach	Length of EP Test Set	Length of PUC Test Set	Events Reused	Generation Time (ms)	Faults Seeded	Events Executed	Faults Revealed
BA	8	*inc*	21	25	21	0.94	22	77	16
		sm	-	25	-	71.48		83	17
eM	7	*inc*	27	36	29	1.07	124	213	53
		sm	-	33	-	55.59		210	53
SAS	2328	*inc*	128	133	128	0.88	251	943	154
		sm	-	128	-	76.54		920	149

used PUCs[6] and the selected testing scenarios[7] are given in external resources.

D. FAULT SEEDING

From an event-based view, a fault is observed in the form of an event that is either missing or extra after a particular (m-1)-sequence (m⩾2) and, by increasing m, the fault domain can be extended [8]. Both *inc* and *sm* target missing event faults associated with 2-sequences; that is, an event does not follow after a particular event although it should. Thus, in order to gain a more realistic insight into testing performances of *inc* and *sm*, missing-event faults associated with m-sequences where m = 2,3 are considered. For each PUC, 20% of the total number of possible faults are randomly generated and seeded. Data on the number of possible faults and the number of seeded faults are given in external resources for each SPL and PUC.[8]

E. TEST GENERATION AND EXECUTION

Before we present the details on test generation and test execution processes, we give some implementation and infrastructural details. To obtain test generation and execution results, *inc* and *sm* approaches are implemented using Java programming language.[9] Furthermore, test generations and test executions are carried out on a computer with the Intel Core i7-4720HQ 2.60 GHz CPU, 12 GB DDR3 RAM and, Windows10 64-bit OS.

Table 2 presents exemplary data on fault coverage and performance observed during test set generation and test execution processes, that is, the approaches using which PUC tests are generated, the length of each test set (the sum of the lengths of all test sequences in each test set), the number of events reused by *inc* test sets, the time it takes to generate each test set, the number of events executed using each test set and the number of faults revealed using each test set. The complete tables are given in external resources.[10]

F. INTERPRETATION OF THE RESULTS

In this section, research questions RQ1-RQ3 defined in Section IV-B are answered. For questions RQ1, RQ2 and RQ3, we use statistical hypothesis testing. Since we have 30 testing scenarios selected uniformly at random for each of the three SPLs, we take the advantage of the Central Limit Theorem [27] and perform parametric tests [28], [29]. More precisely, we perform paired two-sample t-tests to compare test set sizes, fault detection effectiveness, and cost effectiveness of *inc* to those of *sm*. Furthermore, for question Q1, we analyze the test sets generated by *inc* to see to which level the existing test sets have been reused in the test sets generated by *inc*.

1) RQ1: REUSE LEVEL AND SIZE DIFFERENCES

Let T_e be the existing test set (of an existing product), T_{inc} be the test set of a new product generated using *inc*, *len*(T) be the total number of events in T, and *mpl*(t, T) be the length of the longest prefix of t which is also a prefix of a test case in T. Reuse level of *inc* is defined as follows.

$$RL_{inc} = \sum_{t \in T_e} mpl(t, T_{inc})/len(T_e) \quad (1)$$

(1) is an indicator of how much of T_e's events are reused in T_{inc}. For each testing scenario, RL_{inc} is computed. Our results show that RL_{inc} value is always 100.00%. Existing test sets are completely reused.

To compare the size differences of *inc* and *sm* test sets, let T_{sm} to be the set of test cases of a new product generated using *sm*. One can view Table 3 to see the number of testing scenarios where $len(T_{inc}) > len(T_{sm})$, $len(T_{inc}) < len(T_{sm})$ and $len(T_{inc}) = len(T_{sm})$ for each SPL. In the table, averages of $len(T_{inc})$ / $len(T_{sm})$ are also given.

To decide whether or not the differences between $len(T_{inc})$ and $len(T_{sm})$ values observed in different testing scenarios are significant, we use length per seeded fault (LPSF), which is the ratio of the length of a test set to the number of seeded faults, to normalize the values observed in different testing scenarios. We perform a paired two-sample t-test to see whether the hypotheses that there are no significant

[6]BA https://github.com/esg4aspl/SPL-FESG-Examples/blob/master/BankAccount/Products.pdf
eM https://github.com/esg4aspl/SPL-FESG-Examples/blob/master/Email/Products.pdf
SAS https://github.com/esg4aspl/SPL-FESG-Examples/blob/master/StudentAttendanceSystem/Products.pdf

[7]https://github.com/esg4aspl/Incremental-Testing-in-SPLs/blob/main/IncrementalTestingData/TestingScenarios.pdf (BA: pp.1; eM: pp.2; SAS: pp.3)

[8]https://github.com/esg4aspl/Incremental-Testing-in-SPLs/blob/main/IncrementalTestingData/DataOnNumberOfFaults.pdf (BA: pp.1; eM: pp.2; SAS: pp.3)

[9]https://github.com/esg4aspl/fesg-engine

[10]https://github.com/esg4aspl/Incremental-Testing-in-SPLs/blob/main/IncrementalTestingData/DataOnTestGenerationAndExecution.pdf (BA: pp.1; eM: pp.2; SAS: pp.3)

differences between the LPSFs of *inc* and *sm* test sets can be rejected or not.

Significance values (p values) calculated for paired two-sample t-tests are given in Table 3. **Bold** entries correspond to the tests where the hypothesis is rejected at alpha=0.05 significance level. According to the test results, *sm* test sets are smaller than *inc* test sets for eM and SAS SPLs, and they have similar sizes for BA SPL. This can be explained as follows: ESG-based test generation to cover event pairs is an optimized test generation. However, in the optimization process, pseudo start vertices and pseudo finish vertices are also included. This actually introduces some redundancy to the generated test sets. In test sets of existing products in BA SPL, this redundancy is greater because the product ESGs are relatively smaller when compared to those of eM SPL and SAS SPL. Hence, *inc* manages to reduce this redundancy better than it does in other SPLs and, therefore, sizes of *inc* test sets become similar to those of *sm* test sets.

Also, when we consider the observations obtained using all testing scenarios for all SPLs, *inc* test sets are on the average 9.00% larger when compared to *sm* test sets.

2) RQ2: FAULT DETECTION EFFECTIVENESS

Fault detection effectiveness is evaluated based on number of revealed faults. Let RF_{inc} and RF_{sm} be the numbers of fault revealed by *inc* and *sm*, respectively. To see the number of testing scenarios where *inc* reveals greater number of faults (denoted by $RF_{inc} > RF_{sm}$), *sm* reveals greater number of faults (denoted by $RF_{inc} < RF_{sm}$) and they tie (denoted by $RF_{inc} = RF_{sm}$) for each SPL, one can view Table 4 where averages of RF_{inc} / RF_{sm} are also given.

To decide whether or not the differences between RF_{inc} and RF_{sm} values are significant and we use the fault detection ratio (FDR) which is the ratio of the number of revealed faults to the number of seeded faults. A paired two-sample t-test is performed for each SPL to see whether the hypotheses that there are no significant differences between the FDRs of *inc* and *sm* test sets can be rejected or not.

Significance values (p values) calculated for paired two-sample t-tests are given in Table 4. **Bold** entries correspond to the tests where the hypothesis is rejected at alpha=0.05 significance level. According to the test results, *inc* is more effective at fault detection for BA SPL, and *inc* and *sm* are equally effective for eM and SAS SPLs.

inc manages to reveal a greater number of faults than *sm* in BA SPL, because it manages to cover a significantly greater number of 3-sequences than *sm* does. However, in eM and SAS SPLs, *inc* and *sm* cover a similar number of 3-sequences.

Also, when we consider the observations obtained using all the testing scenarios for all SPLs, *inc* detects 1.33% more faults on the average when compared to *sm*.

3) RQ3: COST EFFECTIVENESS

One of the measures which is used to evaluate cost effectiveness is test generation time. In all testing scenarios, test generation times of *inc* (TGT_{inc}) are smaller than those of *sm* (TGT_{sm}). The average of TGT_{inc} / TGT_{sm} values is also given in Table 5 for each SPL.

TABLE 3. Comparative data on testing scenarios w.r.t. test set sizes.

SPL	$len(T_{inc})>$ $len(T_{sm})$	$len(T_{inc})<$ $len(T_{sm})$	$len(T_{inc})=$ $len(T_{sm})$	$len(T_{inc})/$ $len(T_{sm})$ avg.	p values
BA	1	8	21	0.992234	0.211480
eM	28	0	2	1.242872	**0.000000**
SAS	9	0	21	1.024665	**0.003934**

TABLE 4. Comparative data on testing scenarios w.r.t. revealed faults.

SPL	$RF_{inc} >$ RF_{sm}	$RF_{inc} <$ RF_{sm}	$RF_{inc} =$ RF_{sm}	$RF_{inc}/$ RF_{sm} avg.	p values
BA	5	0	25	1.018771	**0.039217**
eM	9	9	12	1.020261	0.259717
SAS	6	5	19	1.000765	0.813469

TABLE 5. Comparative data on testing scenarios w.r.t. test generation times.

SPL	Average of TGT_{inc} / TGT_{sm}	p Values*
BA	0.021016	**0.000000**
eM	0.020631	**0.000000**
SAS	0.01551	**0.000000**

*All p values are 0.00000 when rounded to 5 decimal places.

To check whether the differences between TGT_{inc} and TGT_{sm} are significant or not, we normalize the test generation times observed in different testing scenarios by computing the ratio of the test generation time to the number of seeded faults which is the test generation time per seeded fault (TGTPSF). A paired two-sample t-test is performed for each SPL to see whether the hypothesis that there are no significant differences between the TGTPSFs of *inc* and *sm* can be rejected or not.

Significance values (p values) calculated for paired two-sample t-tests are given in Table 5. In all the tests, the hypothesis is rejected at alpha=0.05 significance level. Thus, *inc* requires shorter test generation times than *sm*. Upon examination of the values, we see that *inc* generates test cases significantly faster when compared to *sm*. The speedup of *inc* becomes greater from BA SPL to eM SPL and from eM SPL to SAS SPL. This stems from the fact that the sizes of product ESGs increase from BA SPL to eM SPL and from eM SPL and SAS SPL. When we consider the observations obtained using all the testing scenarios, the test generation time of *inc* is 2.86% of that of *sm* on the average.

Another measure to evaluate cost effectiveness is test execution effort which is defined by number of executed events. Let EE_{inc} and EE_{sm} be the numbers of events executed by *inc* and *sm*, respectively. To see in how many testing scenarios *inc* executes greater number of events (denoted by $EE_{inc} > EE_{sm}$), *sm* executes greater number of events (denoted by $EE_{inc} < EE_{sm}$) and they tie (denoted by $EE_{inc} = EE_{sm}$) for each SPL, one can view Table 6. In the table, averages of EE_{inc} / EE_{sm} values are also given.

To see whether or not there are significant differences between EE_{inc} and EE_{sm} values, we use the number of

TABLE 6. Comparative data on testing scenarios w.r.t. executed events.

SPL	$EE_{inc} > EE_{sm}$	$EE_{inc} < EE_{sm}$	$EE_{inc} = EE_{sm}$	EE_{inc}/EE_{sm} avg.	p values
BA	2	18	10	0.924009	**0.000459**
eM	23	4	3	1.087840	**0.000007**
SAS	14	8	8	1.003202	0.532144

executed events per seeded fault (NEEPSF) which is the ratio of the number of executed events to the number of seeded faults. We perform a paired two-sample t-test to see whether the hypothesis that there are no significant differences between the NEEPSFs of *inc* and *sm* can be rejected or not.

Significance values (p values) calculated for paired two-sample t-tests are given in Table 6. **Bold** entries correspond to the tests where the hypothesis is rejected at alpha=0.05 significance level. According to the test results, *inc* require less test execution effort than *sm* for BA SPL, more effort for eM SPL and similar effort for SAS SPL. When we consider the observations obtained using all the testing scenarios for all SPLs, *inc* executes a 0.47% greater number of events on the average when compared to *sm*.

Our findings on numbers of executed events are not parallel to the ones on test set sizes, except for eM SPL where both test set sizes and the number of executed events of *inc* are greater than those of *sm* in general. In BA SPL, test set sizes of *sm* are similar to those of *inc*; however, numbers of events executed by *sm* are greater than those of *inc* in general. Also, in SAS SPL, test set sizes of *sm* are smaller than those of *inc*; however, numbers of events executed by *sm* are similar to those of *inc* in general. This stems from the fact that some relatively long test cases generated by *sm* reveal multiple faults and, each time such a test case reveals a fault, this fault is corrected, and the test case is re-executed from the beginning until all the events in the test case are successfully executed. Our results suggest that these re-executions are more frequent in *sm* when compared to *inc*.

G. THREATS TO VALIDITY

It is always possible to obtain stronger results using a greater number of SPLs. Due to the lack of a repository of event-based models for SPLs and specifications to create event-based models of SPL features, we use three SPLs of different sizes by discerning the functional behavior of their features with the help of other publications, source codes of product implementations and our experience. Also, to minimize this threat, to prevent bias, and to use a statistically sound methodology, we use 90 testing scenarios which are selected uniformly at random, having 30 testing scenarios for each SPL. These scenarios represent cases where incremental testing makes sense; that is, the new product is obtained by adding a new feature to an existing product. In total, 69 different existing products and 79 different new products are used. These products are of varied sizes, especially for different SPLs. Thus, our testing scenarios represent a wide range of different typical situations.

The choice of 30 (the number of testing scenarios for each SPL) may seem arbitrary and insignificant. However, in the theory of probability and statistics, a sample size greater than or equal to 30 is often considered sufficient for Central Limit Theorem to hold [27]. For such samples, parametric tests, which are stronger than their nonparametric counterparts, can be used for comparisons and significant results can be obtained.

Since we promote reuse and the presented approach is evaluated using small increments, each new product is obtained by adding one feature to its existing product in each testing scenario. However, the number of new features is not the best indicator of increment size. There are many additional factors which affect the size of an increment (size of the event-based model of each feature, the rate at which the fault domain expands for each feature, etc.), and, thus, it is very hard to control the increment size. For this reason, we use a simple indicator.

In the evaluations, we generate and seed certain model-based faults (missing-event faults) because these faults are targeted by the considered approaches. There is no proof that the evaluations using model-based faults are relevant for real-world faults, but there is strong evidence supporting that a test set that detects more model-based mutants also detects more code-based mutants [30] and that a test set that detects more code-based mutants also detects more real-world faults [31]. Thus, the evidence suggests that a test set that detects more model-based mutants also detects more real-world faults.

There is no study defining which event-based faults are more common than the others in practice. Therefore, relevant faults associated with m-sequences m=2,3 are generated and seeded randomly to avoid bias. In addition, there is also no work on what number of faults should be used to perform realistic testing evaluations. Similar works use mutation analysis to evaluate test sets. However, mutation analysis does not give any idea of test execution costs. Thus, for the purpose of obtaining realistic results, we randomly select a ratio of all possible faults. We want to keep the ratio of the number of faults the same for all the PUCs. Furthermore, to be able to make substantial observations, we kept this ratio not very small and not too large. Thus, a fixed percentage (20%) of all possible faults is selected for each product.

V. RELATED WORK

In model-based testing (MBT), the specifications of the software to be tested are defined by a model in accordance with the specification. These models are usually graph-based. Examples of these models can be given as finite state machines [32], [33], petri nets [34], and event sequence graphs [16]. A test generation algorithm that takes this model as input creates a test set using a test selection criterion [35].

Whittaker [36] suggested that models used in MBT could be decomposed or combined and showed that test cases can be generated from partial models or model parts and also from the combined large model, and then compared the

results. El-Far and Whittaker [37] examined the issue of test generation from hierarchical models. They showed how the main finite state machine can be expanded by replacing a state with a finite state machine. They made the definition of a hierarchical finite state machine and discussed test generation from hierarchical finite state machines. Belli et al. [38] proposed a method for test generation from hierarchical models that use event sequence graphs. However, these ideas have not been applied to the MBT of SPLs.

Furthermore, ScenTED (Scenario based TEst case Derivations) is one of the first proposed approaches in MBT of SPLs and it provides reuse of the core assets and components for the reuse of the test cases [39]. CADeT (Customizable Activity diagrams, Decision tables and Test specifications) method is also another important research on MBT of SPLs [40] which produces feature-based test cases using UML use case and activity diagrams. Decision tables are used to model variability and generate test cases. A technique called 150% finite state machines, which employs a superimposed model for the SPL under consideration and includes a coverage-driven SPL test suite generation method is proposed by Cichos et al. [41]. Weissleder and Lackner extended this approach and proposed top-down and bottom-up approaches for model-based testing of product lines [42]. Our *sm* technique uses a single model for each product as in the top-down approach of Weissleder and Lackner with the distinction of our models are ESGs and their FSMs. Our *inc* technique is not related to Weissleder and Lackner's top-down and bottom-up approaches.

An approach that uses SAT-based analysis to generate test inputs for each product in SPL is proposed by Uzuncaova et al. [43]. Incremental refinement of test suites for a particular product variant is enabled by their approach. Another approach that reduces the testing effort through reusing test cases taking advantage of SPL architectures similarities is proposed by Neto et al. [44]. They take FSM deltas to reach the next product variant. Our *inc* technique is different from the delta-oriented test generation approaches of Uzuncaova et al. [43] and Neto et al. [44] in the sense that they utilize repeated extension through FSM deltas whereas we use directly an additional feature and its f-ESG model to reach next product variant.

Other several studies utilized delta-oriented SPL testing. For instance, Lochau et al. [45] proposed an integrated delta-oriented architectural test modeling and testing approach for component as well as integration testing. Their approach is component-based and aimed for integration testing, which has no relation to our proposed approach. Dukaczewski et al. [46] proposed requirements-based delta-oriented SPL testing, which takes requirements into focus and uses them to define deltas. Varshosaz et al. [47] proposed to utilize delta-oriented programming to organize FSM-based test models in an incremental structure. Their approach requires object-oriented classes and their FSMs. These three studies have different research directions than ours.

Petry et al. [48] conducted a systematic mapping study and built a roadmap from 44 selected studies. Some of their results concerning our research are as follows: ''Finite State Machines is the most used model to test SPLs'' and ''Behavioral-based and Scenario-based are the most used models'' [48]. In addition to FSM-based modeling of SPLs, Featured Transition Systems (FTSs) are proposed for a mathematical and compact representation of the behavior of an SPL [49]. Devroey et al. [49] utilized FTSs for test generation for SPL products. They also worked on the coverage criteria of FTSs on SPL products [50]. Their approach is different than our approach because of the use of FTSs and they do not perform incremental testing.

All the studies above are based either on FSMs or on FTSs without explicit mapping between features and FSMs or FTSs. In other words, how a single feature is represented by states, transitions, etc. and how states and transitions representing a single feature are connected to an FSM or to an FTS of a product are not depicted. Since these representations linking features to models and products are important for traceability, in our study all features are represented distinctly, their connection to products is clearly stated and the connections can be formed algorithmically, therefore, automatically. Another novelty of the approach proposed in this paper is that it is not required to start with a base product and incrementally reach others.

VI. CONCLUSION AND FUTURE WORK

This paper proposes an approach to incremental testing of products in software product lines (SPLs) by promoting reuse and substantially extending upon the previous approach [14]. The approach is event-based. Featured event sequence graphs (FESGs) are proposed and used for the behavioral modeling of products. Also, the approach is incremental. A novel test generation algorithm is developed by removing the limitations of the algorithm given in [14]; it uses FESG and test cases of an existing product, and feature ESG of a new feature to generate test cases for a new product. In addition, the approach is compared with the state-of-the-art ESG-based approach [6] by using 90 testing scenarios selected uniformly at random over three SPLs.

The proposed approach differs from the state-of-the-art event-based approach [6] in the sense that it allows the user to model a product based on its features and generate test cases. Thus, one does not need to create a new gigantic single model, but instead, test cases are generated incrementally by using existing test artifacts and the model of a new feature. Similarly, when changes occur in specifications, only the relevant features are updated, and their test cases are generated and composed. Evaluations are performed to analyze the characteristics of the proposed approach, namely *inc*, compared to the state-of-the-art single-ESG-based approach, namely *sm*, by using a total of 90 testing scenarios. According to the results, *inc* achieves the highest reuse level of 100% which means all events in existing tests sets are reused. Also, inc generates test sets 34 times faster,

results in 9.00% larger test sets, reveals a 1.33% greater number of faults, and executes a 0.47% greater number of events when compared to *sm* on the average.

Although *inc* yields larger test sets and executes a greater number of events when compared to *sm* (as expected), the percentage of the difference between the test set sizes is greater than 19 times the percentage of the difference between the number of executed events. Thus, *inc* shows a performance much closer to that of optimal *sm* in terms of test execution. Also, the compromise between the difference between the revealed number of faults and the difference between the executed number of events is a good one.

As future work, the incremental approach can be extended to generate test sets achieving k-sequence coverage ($k \geq 2$). In addition, testing scenarios that include the addition of multiple new features to create a new product can be considered. A complementary approach can also be developed to involve the removal of existing features. Finally, the idea of incremental testing realized in this paper using ESGs can be applied to other types of models with richer semantics and/or expressive power.

REFERENCES

[1] J. Withey, "Investment analysis of software assets for product lines," Softw. Eng. Inst., Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep. CMU/SEI-96-TR-010, 1996.
[2] X. Devroey, M. Cordy, G. Perrouin, E.-Y. Kang, P.-Y. Schobbens, P. Heymans, A. Legay, and B. Baudry, "A vision for behavioural model-driven validation of software product lines," in *Proc. ISoLA*, Crete, Greece, 2012, pp. 208–222, doi: 10.1007/978-3-642-34026-0_16.
[3] J. White, J. A. Galindo, T. Saxena, B. Dougherty, D. Benavides, and D. C. Schmidt, "Evolving feature model configurations in software product lines," *J. Syst. Softw.*, vol. 87, no. 1, pp. 119–136, Jan. 2014, doi: 10.1016/j.jss.2013.10.010.
[4] K. Czarnecki, "Generative programming: Methods, techniques, and applications tutorial abstract," in *Software Reuse: Methods, Techniques, and Tools*. Boston, MA, USA: Addison-Wesley, 2002, pp. 351–352.
[5] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (FODA) feasibility study," Softw. Eng. Inst., Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep. CMU/SEI-90-TR-021, 1990, doi: 10.21236/ADA235785.
[6] F. Belli and C. J. Budnik, "Minimal spanning set for coverage testing of interactive systems," in *Proc. 1st Int. Conf. Theor. Aspects Comput.*, 2004, pp. 220–234, doi: 10.1007/978-3-540-31862-0_17.
[7] F. Belli, M. Beyazıt, and N. Güler, "Event-oriented, model-based GUI testing and reliability assessment—Approach and case study," *Adv. Comput.*, vol. 85, pp. 277–326, Jan. 2012, doi: 10.1016/B978-0-12-396526-4.00006-0.
[8] F. Belli and M. Beyazıt, "Exploiting model morphology for event-based testing," *IEEE Trans. Softw. Eng.*, vol. 41, no. 2, pp. 113–134, Feb. 2015, doi: 10.1109/TSE.2014.2360690.
[9] F. Belli and C. J. Budnik, "Test cost reduction for interactive systems," in *Proc. Sicherheit*, Regensburg, Germany, 2005, p. 12.
[10] F. Belli, C. J. Budnik, and L. White, "Event-based modelling, analysis and testing of user interactions: Approach and case study," *Softw. Test., Verification Rel.*, vol. 16, no. 1, pp. 3–32, Mar. 2006, doi: 10.1002/stvr.335.
[11] A. M. Memon, M. L. Soffa, and M. E. Pollack, "Coverage criteria for GUI testing," in *Proc. 8th Eur. Softw. Eng. Conf. 9th ACM SIGSOFT Int. Symp. Found. Softw. Eng. (ESEC/FSE)*, 2001, pp. 256–267, doi: 10.1145/503209.503244.
[12] A. M. Memon, "An event-flow model of GUI-based applications for testing," *Softw. Test., Verification Rel.*, vol. 17, no. 3, pp. 137–157, 2007, doi: 10.1002/stvr.364.
[13] F. Belli and M. Beyazıt, "A formal framework for mutation testing," in *Proc. 4th Int. Conf. Secure Softw. Integr. Rel. Improvement*, 2010, pp. 121–130, doi: 10.1109/SSIRI.2010.23.
[14] T. Tuglular, M. Beyazıt, and D. Ozturk, "Featured event sequence graphs for model-based incremental testing of software product lines," in *Proc. IEEE 43rd Annu. Comput. Softw. Appl. Conf. (COMPSAC)*, Jul. 2019, pp. 197–202, doi: 10.1109/COMPSAC.2019.00035.
[15] A. Classen, "Modelling and model checking variability-intensive systems," Ph.D. dissertation, Faculty Comput. Sci., Univ. Namur, Namur, Belgium, 2011.
[16] F. Belli, "Finite state testing and analysis of graphical user interfaces," in *Proc. 12th Int. Symp. Softw. Rel. Eng.*, 2001, pp. 34–43, doi: 10.1109/ISSRE.2001.989456.
[17] F. Belli, N. Nissanke, C. J. Budnik, and A. Mathur, "Test generation using event sequence graphs," Inst. Elect. Eng. Inf. Technol., Univ. Paderborn, Tech. Rep. TR 2005/6, 2005.
[18] F. Belli and C. J. Budnik, "Test minimization for human-computer interaction," *Appl. Intell.*, vol. 26, no. 2, pp. 161–174, Mar. 2007, doi: 10.1007/s10489-006-0008-0.
[19] A. V. Aho, A. T. Dahbura, D. Lee, and M. U. Uyar, "An optimization technique for protocol conformance test generation based on UIO sequences and rural Chinese postman tours," *IEEE Trans. Commun.*, vol. 39, no. 11, pp. 1604–1615, Nov. 1991, doi: 10.1109/26.111442.
[20] H. Thimbleby, "The directed Chinese postman problem," *Softw., Pract. Exp.*, vol. 33, no. 11, pp. 1081–1096, Sep. 2003, doi: 10.1002/spe.540.
[21] D. West, *Introduction to Graph Theory*. Upper Saddle River, NJ, USA: Prentice-Hall, 1996.
[22] C. Hierholzer and C. Wiener, "Ueber die Möglichkeit, einen linienzug ohne wiederholung und ohne unterbrechung zu umfahren," *Mathematische Annalen*, vol. 6, no. 1, pp. 30–32, Mar. 1873, doi: 10.1007/BF01442866.
[23] F. Belli, M. Beyazıt, and N. Güler, "Event-based GUI testing and reliability assessment techniques—An experimental insight and preliminary results," in *Proc. IEEE 4th Int. Conf. Softw. Test., Verification Validation Workshops*, Mar. 2011, pp. 212–221, doi: 10.1109/ICSTW.2011.59.
[24] T. Thüm, I. Schaefer, S. Apel, and M. Hentschel, "Family-based deductive verification of software product lines," *ACM SIGPLAN Notices*, vol. 48, no. 3, pp. 11–20, Sep. 2012, doi: 10.1145/2480361.2371404.
[25] S. Apel, H. Speidel, P. Wendler, A. von Rhein, and D. Beyer, "Detection of feature interactions using feature-aware verification," in *Proc. 26th IEEE/ACM Int. Conf. Automated Softw. Eng. (ASE)*, Nov. 2011, pp. 372–375, doi: 10.1109/ASE.2011.6100075.
[26] S. Apel, A. von Rhein, P. Wendler, A. Größlinger, and D. Beyer, "Strategies for product-line verification: Case studies and experiments," in *Proc. 35th Int. Conf. Softw. Eng. (ICSE)*, May 2013, pp. 482–491, doi: 10.1109/ICSE.2013.6606594.
[27] P. Olukanmi, F. Nelwamondo, and T. Marwala, "Rethinking *k*-means clustering in the age of massive datasets: A constant-time approach," *Neural Comput. Appl.*, vol. 32, no. 19, pp. 15445–15467, Oct. 2020, doi: 10.1007/s00521-019-04673-0.
[28] D. J. Sheskin, *Handbook of Parametric and Nonparametric Statistical Procedures*, 5th ed. Boca Raton, FL, USA: CRC Press, 2011.
[29] S. Landau and B. Everitt, *A Handbook of Statistical Analyses Using SPSS*. Boca Raton, FL, USA: CRC Press, 2004.
[30] B. K. Aichernig, H. Brandl, E. Jöbstl, and W. Krenn, "Efficient mutation killers in action," in *Proc. 4th IEEE Int. Conf. Softw. Test., Verification Validation*, Mar. 2011, pp. 120–129, doi: 10.1109/ICST.2011.57.
[31] J. H. Andrews, L. C. Brand, and Y. Labiche, "Is mutation an appropriate tool for testing experiments?" in *Proc. 27th Int. Conf. Softw. Eng.*, 2005, pp. 402–411, doi: 10.1109/ICSE.2005.1553583.
[32] S. Fujiwara, G. V. Bochmann, F. Khendek, M. Amalou, and A. Ghedamsi, "Test selection based on finite state models," *IEEE Trans. Softw. Eng.*, vol. 17, no. 6, pp. 591–603, Jun. 1991, doi: 10.1109/32.87284.
[33] T. S. Chow, "Testing software design modeled by finite-state machines," *IEEE Trans. Softw. Eng.*, vol. SE-4, no. 3, pp. 178–187, May 1978, doi: 10.1109/TSE.1978.231496.
[34] D. Xu, "A tool for automated test code generation from high-level Petri nets," in *Proc. 32nd Int. Conf. Appl. Theory Petri Nets Concurrency*, 2011, pp. 308–317.
[35] H. Zhu, P. A. V. Hall, and J. H. R. May, "Software unit test coverage and adequacy," *ACM Comput. Surv.*, vol. 29, no. 4, pp. 366–427, Dec. 1997, doi: 10.1145/267580.267590.
[36] J. A. Whittaker, "Stochastic software testing," *Ann. Softw. Eng.*, vol. 4, nos. 1–4, pp. 115–131, Jan. 1997, doi: 10.1023/A:1018975029705.
[37] I. K. El-Far and J. A. Whittaker, "Model-based software testing," in *Encyclopedia of Software Engineering*. Hoboken, NJ, USA: Wiley, 2002, doi: 10.1002/0471028959.sof207.

[38] F. Belli, N. Guler, and M. Linschulte, "Does 'depth' really matter? On the role of model refinement for testing and reliability," in *Proc. IEEE 35th Annu. Comput. Softw. Appl. Conf.*, Jul. 2011, pp. 630–639, doi: 10.1109/COMPSAC.2011.17.

[39] A. Reuys, E. Kamsties, K. Pohl, and S. Reis, "Model-based system testing of software product families," in *Advanced Information Systems Engineering*. Berlin, Germany: Springer, 2005, pp. 519–534, doi: 10.1007/11431855_36.

[40] E. Olimpiew, *Model-Based Testing for Software Product Lines*. Fairfax, VA, USA: George Mason Univ., 2008.

[41] H. Cichos, S. Oster, M. Lochau, and A. Schürr, "Model-based coverage-driven test suite generation for software product lines," in *Model Driven Engineering Languages and Systems*. Berlin, Germany: Springer, 2011, pp. 425–439, doi: 10.1007/978-3-642-24485-8_31.

[42] S. Weißleder and H. Lackner, "Top-down and bottom-up approach for model-based testing of product lines," *Electron. Proc. Theor. Comput. Sci.*, vol. 111, pp. 82–94, Mar. 2013, doi: 10.4204/EPTCS.111.7.

[43] E. Uzuncaova, S. Khurshid, and D. Batory, "Incremental test generation for software product lines," *IEEE Trans. Softw. Eng.*, vol. 36, no. 3, pp. 309–322, May 2010, doi: 10.1109/TSE.2010.30.

[44] P. A. D. M. S. Neto, I. D. C. Machado, Y. C. Cavalcanti, E. S. D. Almeida, V. C. Garcia, and S. R. D. L. Meira, "A regression testing approach for software product lines architectures," in *Proc. 4th Brazilian Symp. Softw. Compon., Architectures Reuse*, Sep. 2010, pp. 41–50, doi: 10.1109/SBCARS.2010.14.

[45] M. Lochau, I. Schaefer, J. Kamischke, and S. Lity, "Incremental model-based testing of delta-oriented software product lines," in *Tests and Proofs*. Berlin, Germany: Springer, 2012, pp. 67–82.

[46] M. Dukaczewski, I. Schaefer, R. Lachmann, and M. Lochau, "Requirements-based delta-oriented SPL testing," in *Proc. 4th Int. Workshop Product Line Approaches Softw. Eng. (PLEASE)*, May 2013, pp. 49–52, doi: 10.1109/PLEASE.2013.6608665.

[47] M. Varshosaz, H. Beohar, and M. R. Mousavi, "Delta-oriented FSM-based testing," in *Formal Methods and Software Engineering*. Cham, Switzerland: Springer, 2015, pp. 366–381, doi: 10.1007/978-3-319-25423-4_24.

[48] K. L. Petry, E. Oliveira Jr., and A. F. Zorzo, "Model-based testing of software product lines: Mapping study and research roadmap," *J. Syst. Softw.*, vol. 167, Sep. 2020, Art. no. 110608, doi: 10.1016/j.jss.2020.110608.

[49] X. Devroey, G. Perrouin, and P.-Y. Schobbens, "Abstract test case generation for behavioural testing of software product lines," in *Proc. 18th Int. Softw. Product Line Conf. Companion Volume Workshops, Demonstrations Tools*, 2014, pp. 86–93, doi: 10.1145/2647908.2655971.

[50] X. Devroey, G. Perrouin, A. Legay, M. Cordy, P. Y. Schobbens, and P. Heymans, "Coverage criteria for behavioural testing of software product lines," in *Proc. Int. Symp. Leveraging Appl. Formal Methods, Verification Validation*, vol. 8802, Oct. 2014, pp. 336–350, doi: 10.1007/978-3-662-45234-9_24.

MUTLU BEYAZIT received the bachelor's and master's degrees from the Department of Computer Engineering, İzmir Institute of Technology, and the Ph.D. degree from the Department of Electrical Engineering and Information Technology, University of Paderborn. He was employed as a Research Assistant during his master's study at the Department of Computer Engineering, İzmir Institute of Technology, and during his Ph.D. study at the Department of Electrical Engineering and Information Technology, University of Paderborn. After completing his Ph.D. study, he was employed as a full-time Faculty Member of the Department of Computer Engineering, Yaşar University.

TUGKAN TUGLULAR (Member, IEEE) received the B.S., M.S., and Ph.D. degrees in computer engineering from Ege University, Turkey, in 1993, 1995, and 1999, respectively. He worked as a Research Associate at Purdue University, from 1996 to 1998. He has been with the İzmir Institute of Technology, since 2000. After, he became an Assistant Professor at the İzmir Institute of Technology, where he worked as the Chief Information Officer, from 2003 to 2007. Currently, in addition to his academic duties, he acts as an IT Advisor to the Rector. He has more than 60 publications and active record of duties with international and national conferences. His current research interests include model-based testing and model-based software development.

DILEK ÖZTÜRK KAYA received the B.S. and M.S. degrees in computer engineering from the İzmir Institute of Technology, in 2017 and 2020, respectively, where she is currently pursuing the Ph.D. degree with the Department of Computer Engineering. She is also a Research Assistant with the Department of Computer Engineering, İzmir Institute of Technology. Her current research interests include model-based testing, software product line, and software quality.