Journal of Information Processing Vol.26 572–589 (Sep. 2018)

[DOI: 10.2197/ipsjjip.26.572]

Invited Paper

Model-based Selective Layer-centric Testing

Fevzi Belli[1] Nevin Güler Dincer[2] Michael Linschulte[3] Tugkan Tuglular[4,a)]

Received: May 10, 2018, Accepted: June 18, 2018

Abstract: Model-based testing of large systems usually requires decomposition of the model into hierarchical submodels for generating test sequences, which fulfills the goals of module testing, but not those of system testing. System testing requires test sequences be generated from a fully resolved model, which necessitates refining the top-level model, that is, by replacing its elements with submodels they represent. If the depth of model hierarchy is high, the number of test sequences along with their length increases resulting in high test costs. For solving this conflict, a novel approach is introduced that generates test sequences based on the top-level model and replaces elements of these sequences by corresponding, optimized test sequences generated by the submodels. To compensate the shortcoming at test accuracy, the present approach selects components that have lowering impact on the overall system reliability. The objective is to increase the reliabilities of these critical components by intensive testing and appropriate correction which, as a consequence, also increases the overall reliability at less test effort without losing accuracy. An empirical study based on a large web-based commercial system is performed to validate the approach and analyze its characteristics, and to discuss its strengths and weaknesses.

Keywords: model-based testing, model refinement, event sequence graphs, software reliability, assignment problem, Chinese postman problem

1. Introduction

Model-based testing (MBT) focuses on relevant aspects of the system under consideration (SUC) and generates test cases from the model of the SUC's behavior. MBT approaches enable algorithmic generation, updating, and reuse of sets of test cases, forming test suites. Depending on the complexity of SUC, generating tests from the model can precipitate a state space explosion. Therefore, most of the existing techniques require refinement of the starting model by additional sub-models (representing components of the SUC), resulting in a hierarchical decomposition [2].

Assuming that the model is represented as a directed graph $G = (V, E)$, where V denotes the set of vertices, and E the set of edges (or arcs), test set generation usually requires generation of paths along G. Furthermore, assuming the run-time complexity of finding a minimal test set to cover the edges (as vertex sequences of the length two) of G is $O(|V|^3)$, where $|V|$ denotes the number of vertices [1], it is quite obvious that the test generation effort increases with the increasing model size, that is, the number of hierarchy layers, and the length of vertex sequences to be covered [2]. Depending on the chosen sequence length n to be covered, graphs with $(|V|^{n-1} * (n-1))$ vertices have to be solved in the worst case [1], [2]. For instance, if the model consists of 50 vertices, a graph with up to 25,000,000 vertices has to be solved to reach a minimal coverage of all sequences of the length five.

A test generation method recently introduced could factor in the advantage of the hierarchical structure in order to reduce the effort of test generation [2]. In addition, Belli et al. [3] analyzed the effect of test sequence length on the fault detection capability of model-based testing for a single layer model. It turned out that there is a diminishing return in terms of the additional faults detected by longer sequences; yet, there is some value in the additional faults detected, and it would be desirable to cover longer sequences if the cost could be controlled.

A way out of this conflicting situation is a selective construction of the test suites, that is, not considering all of the components included by the fully resolved (FR) model, but only a subset of it for further testing. All other submodules will be represented by test sequences constructed by unit testing, using graph theoretic optimization techniques [1], [2], [3]. Consequently, such a strategy would lead to a compromising of the thoroughness of test coverage. In this context, the following questions are addressed:

Q1. How can components be selected for more intensive testing that provide a better chance of detecting additional "attractive" faults than others?

To answer Q1, we propose to measure the impact of each component on overall system reliability by determining

(i) the usage ratio (UR) of components,
(ii) the reliability of each component (R_k, k = 1, 2, . . . , number of components),
(iii) combined reliability (R_c) as overall reliability of SUC.

This paper proposes a new technique to calculate R_k.

Q2. What is the impact of the selective layer centric testing strategy on the system reliability?

To answer Q2, the reliability level achieved by using the new selective layer centric strategy is to be compared with the reliability

[1] University of Paderborn, Germany
[2] University of Muğla SıtkıKoçman, Turkey
[3] Andagon GmbH, Cologne, Germany
[4] Izmir Institute of Technology, Turkey
[a)] tugkantuglular@iyte.edu.tr

level provided by the fully resolved model.

In answer to all of these questions, this paper introduces a novel selective layer-centric testing (SLC) approach. In the proposed method, at the first step layer-centric (LC) testing is performed and detected faults are categorized. At the second step, components and their respective layers are selected according to the impact of each component on the reliability of the overall system for further testing, based on the reliability and usage ratio of each component. At the last step, layer-centric testing is re-executed for the critical layers only by increasing the sequence length.

The novelties of our approach are as follows: There is no approach to our best knowledge that calculates the reliability on the basis of a hierarchical model used for testing a given SUC and that uses this kind of information to detect further faults to increase the overall reliability as is common in reliability growth models. Moreover, to our best knowledge, there is no approach comparable to the one presented in this paper for making use of model hierarchy for producing optimized test suites.

Section 2 summarizes related work. Section 3 presents the theory behind and describes LC testing through a running example. Section 4, the core of the paper, describes the SLC strategy for selecting a fault-sensitive subset of components and generating tests for them. Section 5 validates the approach and determines its characteristic features in an empirical study based on a web-based software system. Section 6 concludes the paper with a summary of the results and an outline of the research work planned.

2. Terminology, Related Work

The approach combines techniques from two areas that will be reviewed in the following.

2.1 Software Testing

Numerous monographs are dedicated to software testing; e.g., Mathur [4] systematically reviews and presents existing knowledge whereas Binder [5] summarizes relevant techniques for testing object-oriented systems. The books of Myers and Beizer are well-known as well [6], [7], [8]. A common problem that is described in all the books is the derivation of meaningful test cases. A stringent problem is generation of expected test outcomes, which represents test oracle problem. Very often the usage of models is suggested to fill this gap [9].

A broad variety of formal and informal models exists for testing software as recommended in de-facto standards such as UML [1] or TTCN-3 [10]. Depending on user needs, those models enable a state-based or event-based description of the SUC at different levels of granularity and preciseness. These methods and their features will be summarized in the following.

State-based vs. Event-based models State-based, graphic models [4], [11] have been in use for a long time, e.g., for conformance testing [12], [13] as well as for specification and testing of system behavior [14] [15], [16]. One of the earliest models based on finite state machines (FSM) was introduced by Chow [17]. Soon, event-based models were also introduced, e.g., using event-flow graphs (EFGs) [18], and in a broader sense, event sequence graphs (ESGs) [19]. Although nodes are interpreted in both models as operations of an event set [20], EFGs are primarily designed for GUI modeling. A further difference is that ESGs enable a complementary, analytic view which is necessary to model and algorithmically detect potential user errors and undesirable situations [21], [22].

Test Adequacy A common problem of model-based testing is that a very large number of test cases can be derived from a model. This requires a stopping rule for testing, known as test adequacy criterion, which can also be used to determine the "thoroughness" of the testing process [4]. Apart from several model-specific test selection criteria [23], well-known criteria for graph-based models are coverage of all-nodes and all-edges [24].

Test Sequence Length Also the sequence "length" to be covered has to be taken into account [19]. Arcuri [25] investigated the role played by the length of test sequences, particularly branch coverage, and has empirically shown that longer test sequences can improve the results. Contrarily, Belli et al. [3] showed that the number of faults additionally detected decreases when the length of test sequences is steadily increased. They found that most of the faults are detected by covering event sequences of minimal length; that is, 2. Fraser et al. [50] performed a set of experiments, using a genetic algorithm, on the properties of test sequence length and how to counter the effects of length bloat in the context of branch coverage. Their experiments showed that the success rate and coverage for the same amount of resources are significantly higher when applying the bloat control techniques, which are used in search-based testing for object-oriented software [50].

Test Generation The test generation method for fulfilling the selected adequacy criterion plays an important role in the test process. Jourdan et al. [26] analyzed the lower bounds on lengths of checking sequences for FSMs where it is hard to achieve a guaranteed minimum for test execution. However, algorithms to derive a minimal set of test sequences can often be related to common graph problems, e.g., the Chinese postman problem for covering each edge [27], [28] or the traveling salesman problem for covering each vertex [29]. Under certain circumstances, it is even possible to form the (sub-) problem as a linear program, which can then be solved by the simplex method if a minimum is desired [30]. An example is the assignment problem which has to be solved within the Chinese postman problem although solutions with a better runtime exist [1], [29].

Solutions to the assignment problem [29] attempt to assign n items (agents) to n other items (tasks), in such a way that the total cost of the assignment is minimal. It is known from graph theory [44] that the construction of a minimal set of edges, which creates a Eulerian graph, leads to the assignment problem that can be solved in alternative ways. One of the fastest methods for solving assignment problems is the Hungarian method [29], which provides a solution in $O(n^3)$ time. Apart from the Hungarian method, other $O(n^3)$ solutions are given by Dinic-Kronrod [29] and Cycle Canceling [28].

Model Refinement An interesting question is the role that model refinement, more precisely the "depth" of the modeling or its granularity, plays in MBT. The principle of "divide-and-conquer" is not new; Parnas [31] was already considering hierarchical structures for modularization of computer programs in

Table 1 Classification of some NHPP Models.

Model	Abbr., Ref.	Mean value function	Parameter	Class/Family	
Goel-Okumoto	G-O [60]	$\mu(t) = a(1 - e^{-bt})\ b > 0$	a: expected number of faults b,c: constants; initial values	exponential class	finite failure model
Generalized Goel-Okumoto	GGO [61]	$\mu(t) = a\left(1 - e^{-bt^c}\right)$ $a > 0, b > 0, c > 0$		weibull class	
Delayed S-Shaped	D-S [62]	$\mu(t) = a(1 - (1 + bt)e^{-bt})$ $a > 0, b > 0$	a: expected number of faults b: fault detection rate	gamma class	
Log-Power	LP [63]	$\mu(t) = a \ln^b(1 + t)$ $a, b > 0, t \geq 0$	a: scaling factor b: shape parameter	power family	Infinite failure model
Duane	D [64]	$\mu(t) = a \cdot t^b$			
Musa-Okumoto	M-O [65]	$\mu(t) = \frac{1}{\theta}\ln(\lambda_0 \theta t + 1)$	a: failure intensity decay μ: initial failure intensity	geometric family	

1972. His thesis that "the effectiveness of modularization depends upon the criteria used in dividing the system into modules" is valid also for test case generation from hierarchical graph-based models as practiced by MBT. As an example, Memon et al. [32] use an automatic planning system to generate test cases from GUI events and their interactions called planning assisted tester for graphical systems (PATHs). Paiva et al. [33] presented an approach based on hierarchical FSMs where the hierarchical structure is given special attention during the test case generation process. The structure of hierarchical FSMs is exploited to reduce the number of states in the "flat" finite state machines, thus providing a way to deal with the state explosion problem. Andrews et al. [34] propose a system-level testing technique that combines test generation based on FSMs with constraints. They use a hierarchical approach to model large web applications and use constraints to select a reduced set of inputs to decrease the state space explosion. Reza et al. [35] use hierarchical predicate transition Petri nets to model the behavior of SUC and to generate adequate test cases.

All of the above-mentioned approaches deploy hierarchical structures. However, to our best knowledge, there is no approach comparable to the one the present paper introduces for making use of this hierarchy for producing optimized test suites.

2.2 Software Reliability and Its Modeling

Software reliability (SR) models Since the early seventies of the last century, probabilistic models have been used to determine the software reliability (SR) based on observations obtained from software testing. SR is usually used to decide when to stop testing. Some authors use SR models also to analyze the role of different test sequence lengths with respect to its fault detection effectiveness [3]. In this context, non-homogenous Poisson Process (NHPP) models are good candidates because of their compatibility with real world situations and simplicity of computation. They belong to the class of "reliability growth models" since they assume that faults are incrementally detected by tests and immediately (and perfectly) corrected, thus continuously improving the reliability of the SUC.

Musa-Okumoto (M-O) [65], Goel-Okumoto (G-O) [60], and Delayed S-Shaped (D-S) [36], [62] are well-known NHPP models that are recommended by standards [37], [38], [39]. The critical question when applying a NHPP model is that of determining the appropriate mean value function, which eases the derivation of software reliability. This paper considers NHPP models that follow the Musa-Okumoto classification scheme to cover the different types of the models instead of considering all the numerous existing models [36].

Poisson type models can be classified as Homogenous Poisson Process (HPP) and Non-Homogenous Poisson Process (NHPP). HPP models assume that failure rate does not change during the testing process, in other words, SUC has constant failure intensity. In case, where failure intensity varies with the time parameter since faults are only counted once and it is assumed that no new faults are inserted, NHPP models are favored [2]. Overview of the some NHPP Models is given in **Table 1**.

Component-based SR Reliability can be determined twofold: through (i) system-level reliability estimation for SUC as a whole, and through (ii) component-based reliability estimation using the reliability of the individual components of SUC and their interconnection mechanisms. The following questions thereby arise: How to estimate the reliability of individual components, and how to aggregate and analyze these reliabilities. State-based frameworks for component-based software reliability prediction are available in Ref. [40]. A different approach identifies critical components and investigates the sensitivity of the application reliability with respect to these components [41]. Krishnamurthy et al. [42] assess the reliability of component-based applications based on test information and test cases.

Tyagi et al. [57] focused on four factors that have the strongest effect on component-based software system (CBSS) reliability: two main factors to estimate component reliability; (i) the reusability of the component, (ii) the operational profile for the

component, and two main factors to estimate interface reliability; (iii) component dependency, (iv) application complexity. Based on these four factors, they proposed a new fuzzy-logic-based model for estimating CBSS reliability.

Singh et al. [58] proposed an approach to predict the software reliability by modelling the software system through Petri Net, converting it into Markov chain and solving the linear system mathematically. Li et al. [59] proposed a reliability evaluation model for component-based software systems, which utilizes the complex network theory based on the state-based evaluation approach. In their model, the most influential node discovery algorithm in complex network theory is used to calculate the impact factor of each component and then the reliability of the software system is evaluated based on these impact factors.

Obviously, component-based reliability estimation techniques are promising candidates to be considered for the selective testing strategy presented in this paper because the layers of hierarchical models represent components of the SUC. There is no approach to our knowledge that (i) calculates the reliability on the basis of a hierarchical model used for testing a given SUC, and that (ii) uses this kind of information to detect further faults to increase the overall reliability as is common in reliability growth models. The approach presented in this paper introduces a solution to this problem.

2.3 Selective Testing

Offutt et al. [51] suggested selective mutation testing as a way to approximate mutation testing, so that the number of mutants to be executed is reduced. They found out that selective mutation-adequate test sets are effectively equivalent to non-selective test sets in their power to kill mutants for small programs. Chen et al. [52] proposed a system called TestTube that uses static analysis to perform selective retesting of software. With each new version of software, TestTube determines the entities changed and selects tests that covers those changed entities.

Hirayama et al. [53] proposed a selective software testing based on priorities assigned to functional modules. Their method consists of three steps as follows: (i) assign priorities, which are calculated using product and process properties, to functional modules, (ii) derive test specification, and (iii) construct a test plan according to the priorities. They showed that their method is superior to the conventional testing method. Hirayama et al. [54] introduced test item prioritizing metrics for selective software testing. The priorities used in test selection are determined based on the evaluation of three metrics for functions: the frequency of use, the complexity of use scenario, and the fault impact to users. Through experiments they confirmed that their selective system testing can detect both fatal faults related to key functions as well as critical faults for the system.

Steinert et al. [55] proposed an approach that automatically derives a subset of unit tests based on actual modifications to the code base at hand, then continuously executes them transparently in the background. To determine the subset of unit tests, their approach relies on dynamic analysis using internal program representation available in IDEs. Andrews et al. [56] provided a cost-benefit tradeoff framework to determine whether selective regression testing or brute force regression testing is preferable.

Different than the existing selective testing research, our approach uses component reliability values in selecting tests and re-executing them not using the same length but longer test sequences.

3. Background

This section summarizes layer-centric (LC) test generation technique introduced in previous work [2] in order to briefly explain the background for the proposed approach by a running example. Formal definitions can be found in Refs. [2], [43].

The approach proposed in this paper prefers ESG notation for modeling. This preference causes no loss of generality because ESG, like EFG, is equivalent to FSM as all three can be represented by regular (type-3) grammars and thus can be interchangeably used [20]. The reason of this preference stems from the fact that events are externally perceptible and thus objectively observable phenomena, contrary to "states" that are internal to the SUC. Thus, events enable controllability of the test process [49].

3.1 Event Sequence Graphs

Vertices of ESG represent events, that is, environmental or user stimulus, or system responses punctuating different stages of system activity. Directed edges connecting two events define allowed sequences among these events. Formal definitions and detailed explanations related to ESG can be found in Ref. [43].

A vertex representing a single, self-contained event is called an atomic event/vertex. Alternatively, a vertex can be refined by another ESG as a sub-graph (see **Fig. 1**), the vertices of which can also be refined, resulting in a hierarchy of models. Events that can be refined are compound events/vertices consisting of atomic events and/or even other compound events (for details see Ref. [43]).

A sequence of n + 1 consecutive events that represents the sequence of n edges is called an event sequence (ES) of length n + 1, e.g., an ES of length 2 is an event pair (EP). An ES is complete if it starts at the initial event of the ESG and ends at the final event; in this case, it is called a complete ES (CES). Occasionally, CESs are also called walks (or paths) through the given ESG. Accordingly, a faulty event sequence (FES) of length n consists of n − 2 EPs and ends up with an FEP. An FES is complete if it starts at the initial vertex of the ESG; in this case, it is called a faulty complete ES, abbreviated as FCES. An FCES must not necessarily end at the final event. FESs that are not complete, can be completed by ESs (starters) that begin at the entry and end at the first node of the considered FES (for details see Ref. [43]).

CESs and FCESs form test cases for the SUC. A CES is supposed to lead to the exit vertex. If this is not feasible, the corresponding CES is marked as failed (positive testing, that is, checking whether or not SUC is doing what it is expected to do). In contrast, an FCES is not supposed to lead to the final event since it ends with an FEP, which should not be executable (negative testing, that is, checking whether or not SUC is not doing what it is not expected to do). If this is feasible, the corresponding FCES is marked as failed. Hence, by analyzing ESG models, merely faults on events and their order can be detected. Other types of

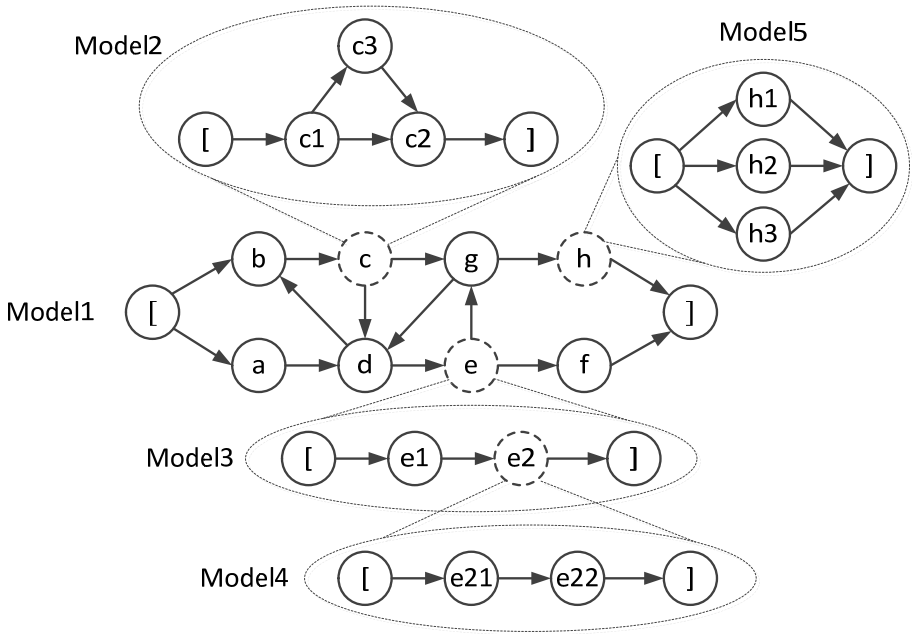


Fig. 1 Refinement of compound vertices.

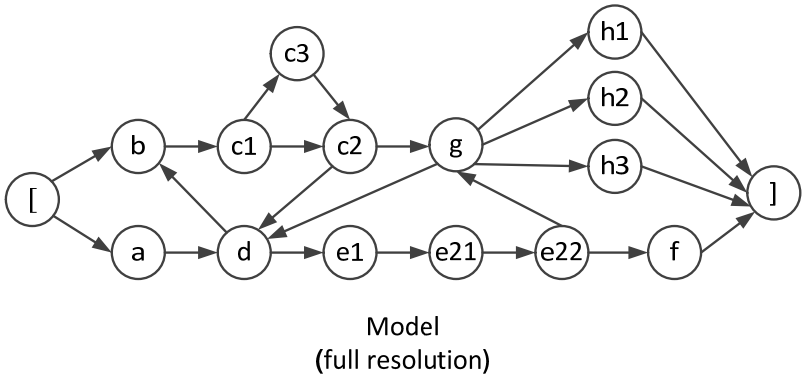


Fig. 2 A full resolved model integrating the models given in Fig. 1 (a *full resolution*).

faults, for example the ones likely in database interactions, are not within the scope of this testing but they might be detected by chance. However, before test sequences can be generated, compound vertices are to be resolved according to ESG definitions (for details see Ref. [43]).

Example 1. In Fig. 1, the refinements of vertices c, e, h of Model 1 are given as Model 2, Model 4, and Model 5, respectively. Since hierarchy of models is allowed in ESG models, there is a refinement also in Model 3, e2 is refined through Model 4. The resolved version of models in Fig. 1 is given in **Fig. 2**. Model 1 is defined as

$V_1 = \{a, b, c, d, e, f, g, h\}$,

$E_1 = \{(a, d), (b, c), (c, d), (c, g), (d, b), (d, e), (e, f), (g, d), (g, h)\}$,

$\Xi(\text{Model 1}) = \{a, x\}$ and $\Gamma(\text{Model 1}) = \{f, h\}$.

(Ξ and Γ denote the start and exit vertices, respectively).

In the refinement, there are four models given in Fig. 1. For instance, the compound event c is resolved by Model 2. The refinement of Model 2 is given by

$V_2 = \{c1, c2, c3\}$,

$E_2 = \{(c1, c2), (c1, c3), (c3, c2)\}$,

$\Xi(\text{Model 2}) = \{c1\}$ and $\Gamma(\text{Model 2}) = \{c2\}$.

The full resolved model shown in Fig. 2 is given as

Model 6 = (V_6, E_6) with

$V_6 = \{a, b, c1, c2, c3, d, e1, e21, e22, f, g, h1, h2, h3\}$ and

$E_6 = \{(a, d), (b, c1), (c1, c2), (c1, c3), (c3, c2), (c2, d), (c2, g), (d, b), (d, e1), (e1, e21), @(e21, e22), (e22, f), (e22, g), (g, d), (g, h1), (g, h2), (g, h3)\}$,

$\Xi(\text{Model 3}) = \{a, b\}$ and $\Gamma(\text{Model 3}) = \{f, h1, h2, h3\}$. □

3.2 Test Case Generation from ESG

CESs and FCESs form the test sequences (test cases). For a thorough positive testing of ESGs, all EPs of a given ESG are to be covered by CESs of minimal total length and/or minimal number. This problem is a derivation of the *Chinese postman problem* that attempts to find the shortest path or cycle in a graph by visiting each arc Ref. [1].

As already mentioned above, hierarchical models are supposed

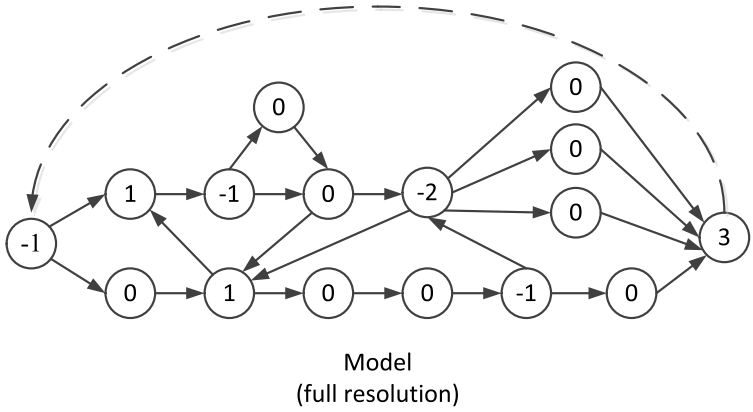


Fig. 3 Degrees of the vertices of the ESG given in Fig. 2.

to be resolved completely before CESs are generated. The run-time complexity of finding a minimal solution is $O(|V|^3)$, where |V| denotes the number of vertices [1]. The number of FCESs for negative testing increases with in-creasing number of vertices since $|FCES| = |V|^2 - |E|$.

Example 2. The technique proposed in Ref. [1], [43] produces the following CESs, which cover all EPs of the given ESG, as test sequences for the model given in Fig. 2:

CES1 = [a d e1 e21 e22 g d e1 e21 e22 f]

CES2 = [b c1 c3 c2 g h1]

CES3 = [b c1 c2 d b c1 c2 g h2]

CES4 = [b c1 c2 g h3]

Below, some of the produced FCESs are presented as negative test cases.

FCES1 = [a a

FCES2 = [a b

The set of negative tests has $|FCES| = |V|^2 - |E| = 14^2 - 17 = 179$ elements. Adding the four positive test cases results to a total of 183 test cases for this example. The set of CESs of Example 2 has been generated by a solution to the Chinese postman problem. □

Figure 3, based on Fig. 2, shows the degrees for each vertex of Fig. 2 with set A = {],],], b, d} and set B = {[, c1, e22, g, g}. The vertex] occurs three times in set A since its degree is +3. The vertex g occurs twice in set B since its degree is −2. Note that an edge is added from end vertex] to start vertex [in Fig. 3 to fulfill the requirement of strong connectivity.

The challenge in deriving the minimal set of edges, i.e., for balancing the graph, is to assign each element of set A to exactly one element of set B so that there is no unassigned element in either set and there is no other assignment with a lower number of edges to be added (according to the assignment). This leads to an assignment problem.

Considering ESGs, the costs are defined by the number of edges of the shortest path between vertex $i \in A$ and vertex $j \in B$, and n is the number of elements of set A or B, respectively. An assignment of vertex i to vertex j indicates that edges along the shortest path from vertex i to vertex j have to be added. Note that set A and B should have the same size since the sum of all degrees in a given graph is zero; i.e., $\sum_{v \in V} \delta(v) = 0$ and hence, n = |A| = |B|.

Table 2 The Resulting Cost Matrix out of Fig. 3.

c_{ij}	**[**	**c1**	**e22**	**g**	**g**
]	1	3	6	5	5
]	1	3	6	5	5
]	1	3	6	5	5
b	6	1	6	3	3
d	6	2	3	4	4

Example 3. **Table 2** shows the resulting cost matrix to be solved for Fig. 2. The matrix elements grade the minimal number of edges (as costs) if a node represented by row i is assigned to (connected with) a node represented by column j. The goal is to find an assignment of row i to column j so that each row i is assigned exactly once to one column j, and each column j is assigned exactly once to one row i. A minimal assignment is indicated by dark grey boxes in Table 2. Furthermore, there should be no other assignment with a lower sum of costs. According to Table 2, the following shortest paths must be added to the ESG given in Fig. 2 to create a minimal Eulerian cycle:] → [,] → g, $b \rightarrow c1$, $d \rightarrow e22$. □

3.3 Layer-centric (LC) Testing and Its Reliability Analysis

The new strategy introduced in Section 4 is built up on top of an existing strategy [2], which is briefly summarized in this section. The ESG notions and graph-theoretic results introduced above help to address the question: "*How can the effort of test generation as well as the excessive number of test cases be reduced?*"

The basic idea for solving the problem of increasing complexity endemic to resolving the hierarchical structure is to generate test cases for each ESG individually, which is called layer-centric testing [2]. This reduces the effort of finding a minimal solution since $O(|V_{resolved}|^3) > O(|V_1|^3) + \cdots + O(|V_k|^3)$, where k is the number of single ESGs forming the hierarchy that is, $k = 5$ for Fig. 1. This strategy will also reduce the number of negative test cases significantly since FEPs between different ESG layers are not considered. As a consequence, faults occurring between different layers cannot be detected. But generating test cases for

Table 3 Number of CESs and FCESs for Models in Fig. 1.

	Model 1	Model 2	Model 3	Model 4	Model 5
Number of CES	2	2	1	1	3
Number of FCES	54	6	3	3	9

each ESG on its own also introduces several problems for test generation, which is discussed below.

Problem 1: Effect of Compound Vertices on Test Generation

Compound vertices, which represent compound events, consist of atomic ones; however, their influence on test generation is not clarified.

Example 4. Consider Model 1, Model 2, Model 3, Model 4, and Model 5 of Fig. 1. The optimization algorithm given in Ref. [1], [43] generates the following CESs for five models of Fig. 1.

Model 1:

T1 = [a d b c d e g d e f]

T2 = [b c g h]

Model 2:

T3 = [c1 c2]

T4 = [c1 c3 c2]

Model 3:

T5 = [e1 e2]

Model 4:

T6 = [e21 e22]

Model 5:

T7 = [h1]

T8 = [h2]

T9 = [h3] □

For positive testing, 9 test cases in total are generated (instead of 4, Example 2). The number of the resulting FCESs (negative testing) for Models in Fig. 1 are given in **Table 3**. Compared to Example 2, the total number of test cases has been reduced from 183 to 84.

Analysis of Example 4 reveals the following problem: Compound vertices, e.g., e in Example 4, have more nodes than the atomic ones do. This implies, if there are many test sequences that include compound vertices, test length will very likely increase, and, accordingly, test costs will increase. Therefore, there is a need to determine the weight of the compound events based on the number of atomic events they include.

Problem 2: Executing Compound Vertices

The next problem to be considered is: How can test sequences be executed that contain compound vertices? An example of this problem can be seen in test case T3 of Example 6, where vertex *b* represents a compound vertex.

A straightforward strategy is to replace the compound vertices by test case(s) generated from the lower-layer ESG. If this lower-layer ESG also contains compound events, one has to move down to the next lower-layer ESG, etc., and propagate test cases generated in these layers to upper layers.

Example 5. Since the weight of a compound vertex is given by the length of the shortest CES [2], the weight of Model 2 of Fig. 1 is 2 because the shortest CES possible is [c1 c2]. The weight of Model 3 of Fig. 1 is 3 because the shortest CES possible is [e1 e21 e22]. Note the recursive nature of weight calculation. □

Example 6. If the weight of the compound event is taken into account, the test cases T1 and T2 are modified as follows:

T1 = [a d b c d e g d e f]

T1′ = [a d b c d e1 e21 e22 g d e1 e21 e22 f]

T2 = [b c g h]

T2′ = [b c1 c2 g h3]. □

Problem 3: Executing Lower Layer Test Cases

T1′ and T2′ can be executed using Model 1 at the top layer when considering Example 6. T4, T7, and T8 are to be executed using Model 2 and Model 5 at the next lower layer, which is not desirable. In a potential solution the compound vertices c, e, and h have to occur in the minimal coverage of Model 1 at least as many times as its atomic refinement has test cases. Model 2 has two test cases, namely T3 and T4. Therefore, the solution has to contain at least two occurrences of c. The same procedure has to be repeated for h with at least three occurrences and for e with at least one occurrence. These requirements can be fulfilled through an extension of the assignment matrix of Model 1 by adding columns and rows for the vertices as multiple times needed. The final test case set is:

T1 = [a d e1 e21 e22 g d e1 e21 e22 f],

T2 = [b c1 c3 c2 g h1],

T3 = [b c1 c2 d b c1 c2 g h2],

T4 = [b c1 c2 g h3].

Algorithm 1 describes the LC testing approach for positive testing. This algorithm differs from the former one [1] in that it generates CESs for each ESG on its own instead of resolving the set of hierarchical ESGs. The runtime complexity depends mainly on balancing the corresponding ESG, which is $O(n^3)$, according to the Hungarian method. However, since this algorithm generates CESs for each ESG on its own, it has a better runtime complexity than solving the fully resolved ESG since $O(|V_{resolved}|^3) > O(|V_1|^3) + \cdots + O(|V_k|^3)$, where k is the number of ESGs forming the hierarchy.

Algorithm 1.

Determination of CESs for a Hierarchical ESG According to LC.

Input: ESG = (V, E) with ε = [, γ =] (denoting the start and exit vertices)

Output: a set of CESs

FOR EACH compound event of ESG **DO**

set weight of compound event in ESG;

generate CESs for the corresponding ESG' of the compound event;

balance current ESG considering the generated CESs;

determine CESs on the basis of the balanced ESG; //Euler Tour

replace compound events in the resulting CESs by the CESs of

the compound events; □

In contrast to generating CESs, calculating FCESs for negative testing is considerably easier. FCESs of ESGs of the lower layer are generated first and then returned to the next higher layer, where the shortest path [44] from start vertex [to the corresponding compound vertex $v \in V$ is calculated and concatenated with the given FCESs of the lower layer model. Algorithm 2 generates FCESs. The runtime complexity depends mainly on deriving the shortest path for every vertex. Dijkstra's algorithm can find the shortest path in $O(|V|^2)$. Since the shortest path has to be found for every vertex in the graph, the overall runtime complexity is $O(|V| * |V|^2) = O(|V|^3)$. Algorithm 2 also contains the method for deriving FCESs covering FESs of higher length. A detailed description of deriving CESs covering ESs of higher length can be found in Ref. [2].

Algorithm 2.
Determination of FCESs for a Hierarchical ESG according to LC.
Input: a weighted ESG = (V, E) with ε = [, γ =] (denoting the start and exit vertices) and the desired length of FESs to be covered
Output: a set of FCESs covering faulty event sequences of length
FOR EACH compound event of ESG **DO**
generate FCESs for the corresponding ESG' of the compound event;
FOR EACH FCES **DO**
prepend shortest path from start vertex [to compound event;
set up all FESs of length;
FOR EACH FES **DO**
prepend shortest path from start vertex [to first event of FES;
FOR EACH compound event in FES **DO**
IF compound event is last vertex **THEN**
replace with one of the start vertices of ESG' of the compound event;
ELSE
replace with shortest path through ESG' of the compound event; □

4. Selective Layer-Centric Testing (SLC)

This section introduces the novel strategy to answer to the first question raised in Section 1: *How can components (or sub-models, respectively) be selected for more intensive testing that provide a better chance of detecting additional "attractive" faults than others?*

The conclusion that can be drawn from Section 3 is that the test exhaustiveness and test execution effort can be controlled by appropriate selection of

(i) the ES length to be covered, and
(ii) the strategy for handling model refinement.

The higher the chosen sequence length, the more exhaustive the testing of the underlying SUC, but then the test execution effort is also higher. ESGs allow the generation of a very large set of CESs for testing a given SUC by simply increasing the considered event sequence length step by step, whereby the test effort increases with every step, usually exponentially. Unfortunately, the chance to detect additional faults decreases with increasing sequence length since most of the faults have been already detected by test cases covering sequence length 2 [3]. However, testing event sequences of higher length makes it possible to detect critical faults that can only be detected in specific contexts. Thus, there is a need to detect those critical faults with less testing efforts. Assuming that critical faults are to be detected only by a subset of the models forming the hierarchy, it should be possible to increase testing efforts based on this subset of models only. But first, it is necessary to identify this subset of models that are expected to have a higher fault detection capability.

The approach introduced in this paper assumes that components, which are executed often and are of low reliability degree, usually reveal critical faults. Based on this assumption, **Fig. 4** gives a summary of the resulting steps to be performed for *selective layer-centric* (SLC) testing strategy, which will be explained in detail in the following subsections.

4.1 Step 1: Perform Layer-Centric Testing and Categorize Observed Faults

The basis for an ESG-based test process forms a set of test cases that covers at least event sequences of length 2 (that is, EPs and FEPs for positive and negative testing, respectively). Positive testing checks the conformity of expected behavior of SUC, whereas negative testing is performed to check the SUT behavior in unexpected, undesirable situations [48]. This set of test cases also forms the basis for analyzing which component is likely to conceal additional faults.

For further analysis, it is necessary to execute the underlying test case set covering all EPs and collect the results of their execution. On the basis of the results, detected faults are to be categorized along the given components (represented by ESGs). This is done by identifying each event, which cannot be executed in a CES, and assigning this fault to the corresponding component. The same is performed for each FCES that detects a fault. Similarly, the event, which cannot be executed, along with the corresponding fault is assigned to the appropriate component. The result of this categorization is a number of faults detected by each component or ESG.

4.2 Step 2: Select Layers for Further Testing

To identify the component(s) that most endanger(s) the system reliability, firstly, the reliability and usage ratio of each component have to be calculated separately. Furthermore, the impact of each component on the overall system has to be determined.

Step 2.1: Determining Usage Ratio (UR) for each Component

The Usage Ratio (UR) [45] considers the fact that the components are tested with different efforts during testing. Since a single test case may contain events of different components (test cases are merged during LC testing), it is no longer sufficient to use the number of test cases for reliability calculations. This explains why UR parameter represents the ratio of number of events of each component over the number of events of the overall SUC.

$$UR_k = \frac{E_k}{TEOS} \tag{1}$$

where E_k is number of events of k-th ESG/component and TEOS is the total number of events of the overall test case set.

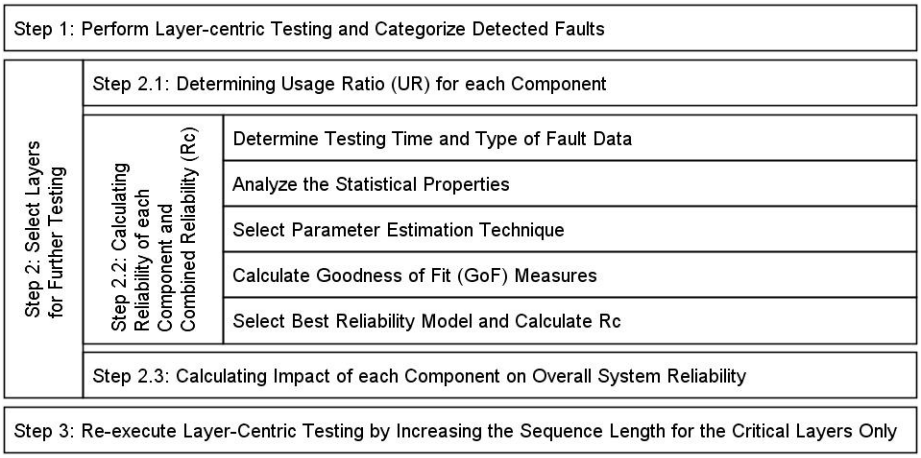


Fig. 4 Summary of the "SLC strategy".

Step 2.2: Calculating Reliability (R_k) of Each Component and Combined Reliability (R_c)

As each of the faults shall be counted only once, assuming they were corrected upon detection without causing new faults, software reliability growth models (SRGM) are used assuming that the absolute number of faults remaining in the software decreases and thus SR grows. Calculating the reliability of each component (R_k) follows similar steps as does the one for calculating the overall system reliability (R_c) [36].

Determine Testing Time and Type of Failure Data

There are several ways to measure test time during the testing process, such as calendar time, number of test runs, and number of test cases or execution time. Moreover, there are two types of fault data for SRGMs: time intervals between successive observed faults and the number of faults detected in a specified time interval. Here, the cumulative number of events generated for each component is used as time parameter. Fault data type is the cumulative number of faults detected in each component. To construct fault data, components are firstly sorted in descending order in accordance with their URs.

Analyze the Statistical Properties

Statistical properties of fault data are analyzed subject to different aspects, for example, whether they follow a specified probability distribution, or whether they form a specific stochastic process. One-Sample Kolmogorov Smirnov Test (K-S) [46] is one of statistical nonparametric tests used to determine whether a sample (failure data collected) fits the specified distribution. As presented in the empirical study, it is observed that the cumulative number of faults builds up Poisson distribution according to K-S test (Section 5.3, **Table 8**). As the exact nature of fault data is not known a priori (except that it can be described as NHPP), several NHPP models must be selected to ensure covering each type (see Table 1).

Select Parameter Estimation Technique

For estimating parameters of SRGMs, Maximum Likelihood Estimation (MLE) technique [46] is used because MLE fulfills most of the favored properties, such as asymptotic normality, robustness and consistency. Besides, MLE simultaneously estimates model parameters and provides to easily de-rive confidence intervals.

Calculate Goodness of Fit (GoF) Measures

GoF measures describe how well SRGMs fit a set of observations. Therefore, GoF measures can also be used to compare different SRGMs according to their correlation to failure data. In this study, Akaike Information Criteria (AIC) and Bayesian Information Criteria (BIC) are used since these criteria are based on the maximized value of likelihood. In addition, commonly used Mean Square Error (MSE) is selected [47].

$$AIC = -2LLF + 2k \tag{2}$$

$$BIC = -2LLF + kln(n) \tag{3}$$

$$MSE = \sum (y - \hat{y})^2/(n - k) \tag{4}$$

where k is number of the model parameters, n is number of observation (number of components), LLF is the log likelihood, y is the observed value, and $\hat{y}$ is the predicted value.

Select Best Reliability Model and Calculate R_c

The SRGM with the smallest AIC, BIC, and MSE are selected as best fitting model. Then, REs are determined according to the best fitting SRGM model as follows:

$$R_k = e^{-\mu(t_k)+\mu(t_k-\Delta t)} \tag{5}$$

where R_k represents the reliability of k-th component of sorted components, t_k is equal to the number of events generated for k-th component, Δt is a small-time interval defined by the user. However, it can be selected as the minimum number of events from among the number of events generated for components. The combined reliability R_c is then defined as follows.

$$R_c = 1 - \sum_{k=1}^{m} (1 - R_k) U R_k \tag{6}$$

where R_k represents reliability of k-th component and m is the number of components.

Step 2.3: Calculating Impact of Each Component on Overall System Reliability

The impact of a component (EI) [45] on overall system reliability can then be determined as follows.

$$EI_k = 1 - \frac{(1 - R_k) U R_k}{1 - R_c} \tag{7}$$

where EI_k represents the impact of k-th component on overall system reliability. Note that components with a low EI value have a higher (negative) influence on the overall system reliability than those with a higher EI value. Thus, the overall system reliability can be improved by increasing EIs of these components.

Components with a small EI value have a (statistically) higher fault detection capability since they also have a high degree of UR and thus, for further testing they can be considered of having a low reliability degree. Following assumptions are made:

- Test process can be stopped if R_c is satisfactory, e.g., if it has a value higher than 0.95.
- Test process can continue if R_c is not satisfactory by selecting the components that have EI values in lowest 25% of all EI values. In statistics, this definition corresponds to the first quartile. Thus, the 1st quartile of EI values determines the set of components with the worst EI values. The selection is made as follows:
  - EI values of n components are ranked from lowest to highest.
  - 1st quartile is bordered by the EI value of the component at position p = (n + 1)/4. If p is not an integer, e.g., p = 3.5, the EI value at position p is calculated as follows: $IE_p = IE_{p<} + (IE_{p>} - IE_{p<}) * (p - p_<)$ where $p_<$ is the (integer) position before p (that is, p is rounded down to the next integer leading to $p_< = 3$) and $p_>$ is the (integer) position after p (that is, p is rounded up to the next integer leading to $p_> = 4$). Quartiles can be computed by using software packages such as MINITAB (http://www. minitab.com).
- If the 1st quartile approach cannot be considered as adequate to improve R_c, testing can be continued using other criteria, e.g., by starting with the component that has the lowest EI value, and then taking the one with the closest value, etc. until test budget is run out.

4.3 Step 3: Re-execute Layer-Centric Testing by Increasing the Sequence Length for the Critical Layers Only

On the basis of the analysis in Step 2, the test effort is to be increased only for the components which have been identified as putting the overall system reliability most at risk. The open question is: *How can test efforts be increased for these components?*

LC testing generates test sequences for every individual model of a given model hierarchy to reduce the test generation and execution effort compared to the FR approach. But the LC testing approach also enables another method of controlling the thoroughness of the test and test execution effort. Based on LC testing, it is possible to increase event sequence length to be covered only for some selected individual models of the hierarchy. Assuming that a given model hierarchy consists of three models, it is possible to generate longer test sequences for just a subset (one or two) of the models whereas the other models are covered by shorter test sequences. The precondition is that the given SUC is represented by a hierarchical set of models. This strategy leads to selective layer-centric testing (SLC).

4.4 Arising Problems

Applying SLC to real-life SUCs introduces some problems. If tests are generated for the uncritical components as well, this leads to additional test effort since a test case set covering sequence length 2 forms the basis for the identification of critical components. Therefore, tests of the uncritical components are expected to have no additional benefit; that is, they are not likely to detect any additional faults. Hence, it would be worthwhile to generate tests only for the critical components. This, however, leads to the following questions that can easily be answered.

1. *How can test sequences containing compound vertices be executed if no tests have been generated for the compound vertex?* Answer: Since the goal is to minimize the test execution effort, the compound vertices are to be replaced by the shortest path through the corresponding ESG. Note that, if some tests have been generated for the compound vertex, they are to be considered during sequence generation as described for LC testing in Section 4.
2. *How can test sequences of lower layers be executed if no tests are to be generated for upper layers?* Answer: Move them to the upper layer as well; that is, generate test sequences for the upper layer so that lower layer tests can be executed. Further details are given in the following.

Executing Lower Layer Test Cases

The second question above refers to the fact that CESs of lower layers are to be executed somehow in the context of upper layers, even if no tests are generated for the upper layer. To keep the costs as low as possible in such cases, a set of CESs is needed for the upper layer containing the number of compound vertices for which as many tests as needed are generated. Furthermore, this set should not be replaceable by another set of CESs with a lower number of total events.

Example 9. Consider **Fig. 5**, where an ESG is given with refined vertices *c*, *e*, and *h* that are symbolized as dashed circles. Temporarily ignore the dashed lined arc from vertex] to vertex [. The ESGs of the compound vertices c and e have been identified as neuralgic and test sequences (CESs) to cover ESs of higher length have been generated for them. The numbers next to the vertices indicate how many CESs have been generated for them, that is, 2 CESs for vertex *c* and 3 CESs for vertex *e*. Thus, the number indicates how often this compound vertex will be needed in a solution. Here, the goal is to derive one or more CESs, which contain vertex *c* at least twice and vertex *e* at least three times and the total number of events is minimal. □

Solving the Traveling Salesman Problem For Testing Lower Layers

Adding an edge from pseudo end vertex] to pseudo start vertex [(as shown in Fig. 5) produces a strongly connected ESG which helps to derive the required set of CESs. This edge enables the problem to be transferred to determine a minimal tour, which starts and ends at pseudo start [and which visits the compound vertices as much as needed. If every compound vertex needs to be visited only once, this problem forms a derivation of the *traveling salesman problem* (TSP) [29].

The TSP attempts to find the shortest possible tour that visits each entry ("city") of a given list. The underlying assumption is that the pairwise distances are known for the cities, e.g., given as a matrix $C = (c_{ij})$. Considering ESGs, the distances consist of the

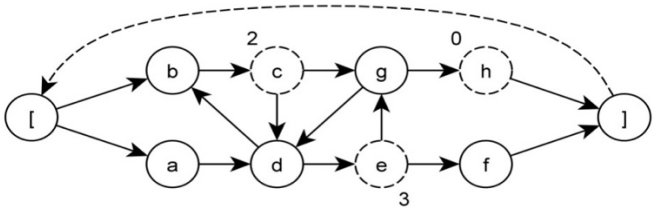
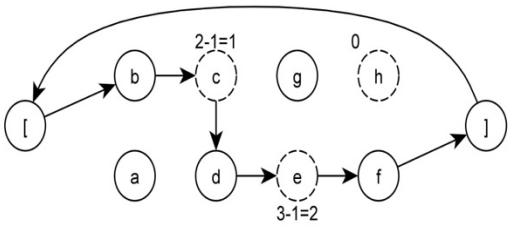
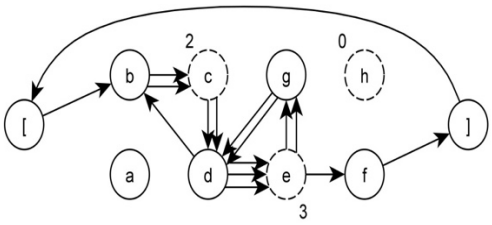


Fig. 5 Example of an ESG with compound vertices *c*, *e*, and *h*.



c_{ij}	[	c	e
[	-	**2**	4
c	4	-	**2**
e	**3**	4	-

Fig. 6 The resulting tour through compound vertices *c* and *e* (above) determined by the solution of the TSP on the basis of the distance matrix (below).



c_{ij}	c	e	e
c	3	**2**	2
e	**4**	3	3
e	4	3	**3**

Fig. 7 The extended ESG (above) along the assignment matrix (below).

minimal number of edges between two vertices. Solving the TSP can also be regarded as solving the AP, but where the resulting assignment needs to describe a cyclic permutation.

Example 10. The right-hand side of **Fig. 6** shows the underlying distance matrix for solving the TSP. The numbers of the matrix represent the minimal number of edges to be visited if a vertex represented by row *i* is assigned to a vertex represented by column *j*. The dark grey boxes indicate the minimal tour; that is, seven edges have to be visited to follow the minimal tour. According to the table in Fig. 6, the following shortest paths must be added to the ESG to denote the shortest tour: [→ c, c → e, e → [. The left-hand side of Fig. 6 illustrates the minimal tour. □

Extending the Tour by Solving the Assignment Problem

On the basis of this initial solution, the next goal is to extend this tour in a minimal way so that the resulting tour contains the desired number of compound vertices needed. This can be achieved by adding this tour into a graph with all edges of the original graph having been deleted (as can be seen in Fig. 6). After that, this graph is to be extended by additional edges so that the resulting Eulerian cycle contains the compound vertices as many times as needed. As already described in Section 3, determining the set of additional edges can be achieved by setting up and solving the assignment problem.

Example 11. As it can be seen in Fig. 6, vertex c is needed one more time in the solution and vertex e is needed two more times in the solution. The right-hand side of **Fig. 7** shows the cost matrix of the corresponding assignment problem to be solved. A minimal assignment is indicated by dark grey boxes. According to this assignment, the following shortest paths must be added to the ESG given in Fig. 6: c → e, e → c, e → e. The resulting ESG can be seen in Fig. 7. On the basis of this graph, the resulting Eulerian cycle starting in pseudo vertex [, looks as follows:

[b c d e g d b c d e g d e f] [□

Note that the last vertex [of the resulting Eulerian cycle does not contribute to the desired result and can be deleted. Furthermore, it might occur that the edge between vertex] and vertex [is traversed more than once in the resulting tour. Thus, the resulting tour is to be split up between the vertices] and [in every place to gain the desired set of CESs.

In contrast to generating CESs, calculating FCESs for negative testing in SLC is considerably easier. As performed similarly in LC, FCESs of selected ESGs of lower layer are generated first and then returned to the next higher layer, where the shortest path [44] from start vertex [to the corresponding compound vertex $v \in V$ is calculated and concatenated with the given FCESs of the lower layer model.

5. Empirical Evaluation

For demonstration and validation of the approach, and analysis of its characteristic features, including a comparison of FR with LC and SLC, a large commercial web application is used. We refrained from a comparison of our approach with random testing because our previous work [3], [43] has already demonstrated the overall effectiveness of ESG approach upon random testing. Instead, in this paper we compare the full resolution and the layer-centric approach with the novel selective layer-centric approach to determine the strengths and weaknesses of the both approaches. The goal is to find out what kind of faults can be detected by our approach and how the detectability changes with varying the test generation method (FR, LC, SLC).

As practiced in real world projects, the testers had learned and understood the system and did not have any prior knowledge about its development to assure the independency between the tester and SUC.

The experiments carried out focus on investigating the third

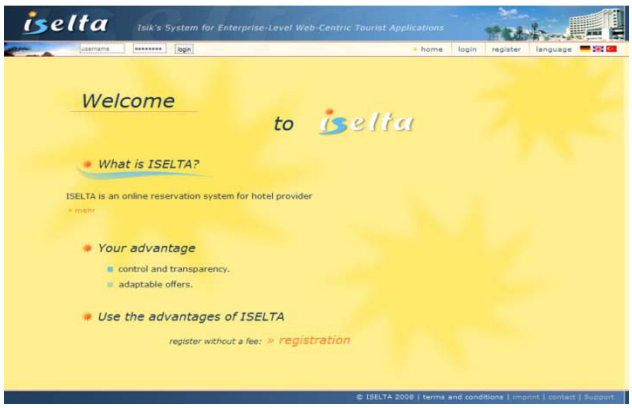


Fig. 8 Screenshot of ISELTA.

question raised in Section 1: *What is the impact of the test case selection process for cost reduction on the overall system reliability?*

5.1 System Under Consideration, Setup and Tool Support

SUC is a large commercial web portal with 53 K LOC (lines of code) called ISELTA ("Isik's System for Enterprise-Level Web-Centric Tourist Applications"). This portal enables travel and tourist enterprises, e.g., hotel owners, to create their own individual search and service offering masks. These masks can be embedded in an existing homepage as an interface between user and system. Customers can then use those masks to select and book services, e.g., hotel rooms, rental cars, etc. See **Fig. 8** for the entry screenshot of ISELTA.

The sub-system *hotel administration* with the following structure has been selected for performing the empirical study.

- *hotel administration* forms a hierarchy of components represented by seven ESGs with a total of 73 vertices and 207 edges.
- More than 60,000 tests have been generated and executed.
- •Three sets of tests, varying the length of ES to be covered (2, 3, 4) have been generated for each of both approaches, FR and LC.

Thus, the chosen sub-system is non-trivial, large, and independent of the others so that it can be viewed as a system on its own. Thus, SUC is an impartially-chosen system. One of the seven ESGs is given in **Fig. 9**.

To support the empirical study, all of the algorithms created for the LC strategy (see Section 3) as well as the SLC strategy (see Section 4) have been implemented and integrated in a tool called Test Suite Designer (TSD) written in Java. TSD automates following steps:

– Modeling of hierarchical components by ESGs via GUI,
– CES and FCES generation following FR, LC or SLC,
– Test script generation for automated test execution.

Vertex annotations of constructed ESGs refer to source code executing the underlying event within a separate test execution environment (see **Fig. 10**). This enables the automatic generation of test scripts along the calculated test sequences. The assumption is that the code snippet can identify the "right" object. Just the name will of course not be sufficient; Object-IDs and other techniques were used to identify objects reliably. In the end, there was

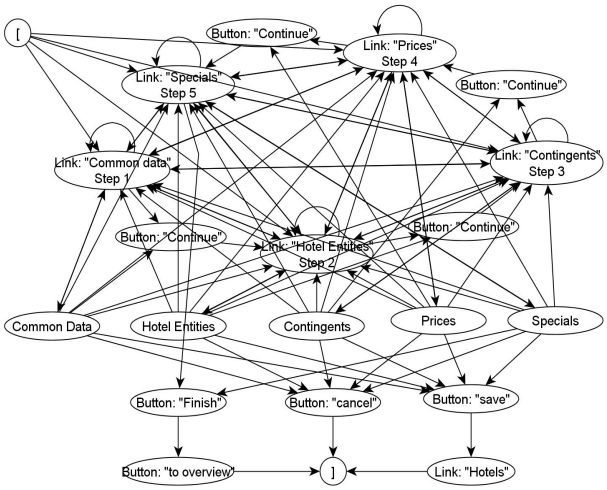


Fig. 9 ESG for modeling the change of hotel data.

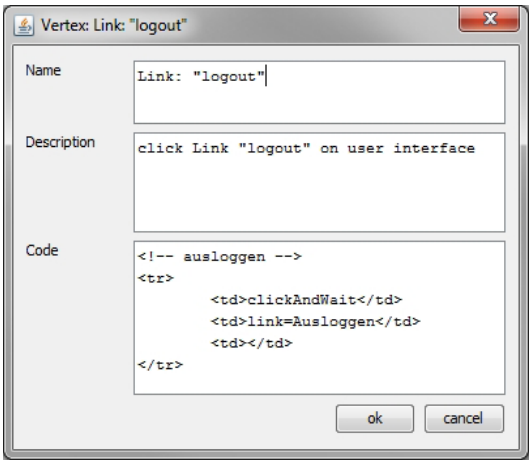


Fig. 10 Entities dialog of an event.

no manual conversion necessary to execute the test which is also one of the strength of the approach. Furthermore, the test oracle defined in Section 3 is easily adaptable to test automation tools like the one used in this study, Selenium (http://seleniumhq.org/).

The comparison of the new approaches, that is, LC and SLC, has been performed in two consecutive steps: First FR with LC, then FR and LC with SLC.

5.2 Comparing FR with LC

Based on TSD, CESs and FCESs for positive and negative testing have been generated and executed along LC and FR covering event sequences of length 2, 3, and 4. The results of the tests are summarized in **Table 4**. It can be seen that the number of CESs in the LC approach is lower than the number of CESs of the conventional FR approach.

To enable a more detailed comparison of the test effort reduction, the number of events to be executed according to each approach is shown in **Table 5**. In total, LC strategy reduced the number of events by 78%. Surprising is the fact that this effort (20% of the original test effort) could detect already 80% of the faults! 31 faults were detected using the LC approach and 39 faults using the FR approach. Thus, LC detected 20% less faults. However, the LC approach reduced the test effort by about 80%. Test execution effort reflected this saving which was reduced from

Table 4 Positive and negative tests subject to ES length.

	FR				LC			
length	CES	FCES	∑	Failures	CES	FCES	∑	Failures
2	5	2668	2673	35	2	585	587	29
3	22	9474	9496	35+4	4	1735	1739	29+2
4	93	38768	38861	39+0	6	7005	7011	31+0
∑	120	50904	51024	39	12	9325	9337	31

Table 5 Number of events to be executed.

	FR			LC			
length	CES	FCES	∑	CES	FCES	∑	Saving
2	353	17504	17857	341	4370	4711	74 %
3	1589	73053	74642	1430	15150	16580	78 %
4	9580	343969	353549	8710	70268	78978	78 %
∑	11522	434526	446048	10481	89788	100269	78 %

4 days to round about one day. The data (in Table 5) show that much of the 74/78% saving is accounted for by reduction in the FCES events. The reduction in CES events is much less: it ranges from 3% to 10%. However, we observed that this saving correlates to the detected faults. It is worth noting that these faults are real faults of the system with respect to the model. They are neither hand-seeded nor resulting from mutation operators.

A reliability analysis (not shown here; however, supplementary material can be provided) compared the reliability of the LC approach (R_{LC} = 0.99940) with the FR approach (R_{FR} = 0.99870) and revealed that the LC approach leads to an even slightly better reliability level than the FR approach ($R_{LC} > R_{FR}$) [2].

5.3 Comparing FR and LC with SLC - Identifying the Critical Sub-Layers for Further Testing

The result of the comparison of LC and FR confirms the result achieved in previous experiments with FR testing (see Ref. [3]). Testing with higher event sequence length also leads here to a great deal of additional test effort while detecting fewer faults.

The question that arises now is: Would it have been possible to achieve a similar reliability level as in LC and FR with less testing effort? To answer this question, the three steps described in Section 4 are carried out.

Step 1: Perform Layer-Centric Testing and Categorize Observed Failures.

Table 6 shows the results of LC testing for each component. The resulting test case set has been analyzed and the events occurring in the resulting test case set have been counted for each of the components. Furthermore, the failures have been categorized along the components. It is assumed now that only the CESs and FCESs covering sequence length 2 based on LC testing have been generated and executed.

Step 2: Select Layers for Further Testing

As mentioned in Section 4.2, the first step to identify a subset of components which have a higher fault detection capability is to calculate usage ratio of each component (UR). Then, the reliability of each component and impact of components on overall system reliability are determined.

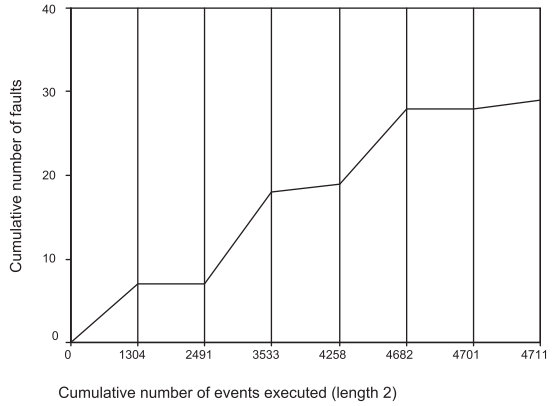


Fig. 11 Failure data used to calculate the reliability of each ESG.

Step 2.1: Calculating the Usage Ratio of Components

Equation 1 is used to calculate the usage ratio of the components. **Table 7** shows the usage ratio of each component. According to Table 7, the component with the highest usage is represented by ESG 4. The component represented by ESG 1 has the lowest usage.

Step 2.2: The New Approach to Calculation of Component Reliabilities

For calculating the reliability of each component, the number of failures observed during the test process and the corresponding number of events (length 2) are sorted in descending order subject to their URs (**Fig. 11**).

To decide whether or not Poisson type models can be used in this study, a K-S test is performed with following hypotheses:

H0: Cumulative number of failures follows Poisson distribution.

H1: Cumulative number of failures does not follow Poisson distribution.

The analysis of the results of the K-S test (Table 8) indicates that the cumulative number of failures follows Poisson distribution (mean parameter = 19.4286) since p-value (0.239) is greater than 0.05.

Table 6 The number of failures categorized according to the number of events.

	ESG1 Main			ESG2 Login Normal			ESG3 Login Quick			ESG4 Prov. Account		
	Events			Events			Events			Events		
Length	CES	FCES	Failures	CES	FCES	Failures	CES	FCES	Failures	CES	FCES	Failures
2	6	4	**1**	5	14	**0**	24	1163	**0**	50	1254	**7**
3	25	13	**1**	12	24	**0**	120	3484	**1**	94	3864	**7**
4	89	34	**1**	22	50	**0**	560	14184	**1**	290	14957	**7**

Table 6 (continued) The number of failures categorized according to the number of events.

	ESG5 Edit Profile			ESG6 Edit Hotel			ESG7 Change Data		
	Events			Events			Events		
Length	CES	FCES	Failures	CES	FCES	Failures	CES	FCES	Failures
2	63	361	**9**	56	669	**1**	137	905	**11**
3	161	1354	**9**	153	2385	**1**	865	4026	**12**
4	543	4241	**9**	911	11565	**1**	6295	25237	**12**

Table 7 Usage ratio of ESGs.

ESG	ESG1	ESG2	ESG3	ESG4	ESG5	ESG6	ESG7
UR	0.0021	0.004	0.252	0.2778	0.09	0.154	0.221

Table 8 One-sample Kolmogorov Smirnov test.

	Cumulative Number of Faults
Poisson Parameter Mean	19.4286
Kolmogorov-Smirnov Z	1.030
p-value (2-tailed)	0.239

The next step is to apply the NHPP models given in Table 1 to the fault data given in Fig. 11 and to compute GoF measures for each SRGM to determine the best fitting model. **Figure 12** shows the results of the GoF measures.

In our previous study, G-O model provided the best performance. However, it can be seen from Fig. 12 that D-S model provides the best performance in all GoF measures since it has the smallest AIC, BIC, MSE values. Therefore, D-S model has been used to calculate the reliability of each component in this study. **Table 9** shows reliability results of each component and combined reliability (R_c).

As can be seen in Table 9, R_c (0.9354) is smaller than R_{LC} (0.99940) and R_{FR} (0.99870) given in Section 5.2. The goal now is to enhance R_c. This will be achieved by performing SLC testing with higher length for components that have a small EI value compared to the overall system reliability.

Step 2.3: Calculating Impact of Component on Overall System Reliability

Table 10 shows the sorted EI values of components on the overall system reliability in line with Eq. (7).

Step 3: Re-execute Layer-Centric Testing by Increasing the Sequence Length for the Critical ESGs Only

SLC testing with sequences of higher length is performed for ESG 3 and ESG 7 (see Table 6) since EI values of these components are equal or less than the 1st quartile of EI values which is calculated as $IE_p = 0.75$ with $p = (n + 1)/4 = (7 + 1)/4 = 2$.

If re-executing LC testing for ESGs in the 1st quartile is not adequate concerning R_c, testing can continue with ESG 4, ESG 6 etc. (see Table 6) respectively. However, there is no need re-executing LC testing for the components that have EI values which are equal or closely equal to 0.9.

In order to demonstrate that the 1st quartile is adequate concerning the reliability requirements for this empirical study, all

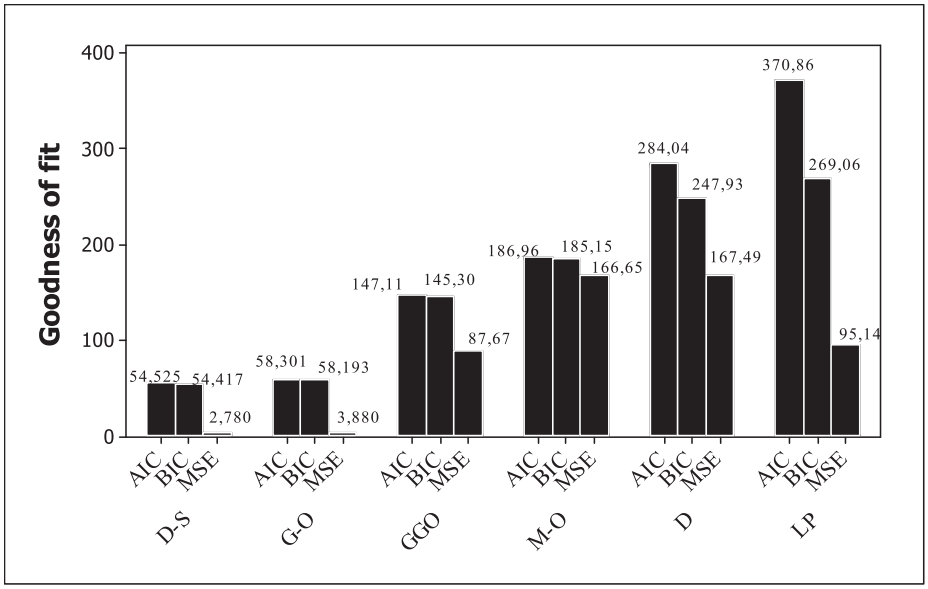


Fig. 12 GoF measures.

Table 9 Reliability results of each ESG and R_c.

ESG	ESG1	ESG2	ESG3	ESG4	ESG5	ESG6	ESG7	R_c
R_k	0.9269	0.9268	0.9329	0.9526	0.9268	0.9261	0.9266	**0.9354**

Table 10 Impact of each ESG on overall system reliability.

ESG	ESG3	ESG7	ESG4	ESG6	ESG5	ESG2	ESG1
Sorted – EI	**0.739**	**0.75 (1st Quartile))**	0.8	0.82	0.9	0.995	0.997

Table 11 The new combined reliabilities calculated by removing ESGs step by step and changes in Model Parameters.

ESG	R_c	D-S Model Parameters		CP	
		a	b	a	b
All ESGs (equals LC-Testing with length 2+3)	0.998	31	0.0006	0	%36.84
Not ESG 1	0.998	31	0.0006	0	%36.84
Not ESG 1+ESG 2	0.998	31	0.0006	0	%36.84
Not ESG 1+ESG 2+ESG 5	0.996	31	0.0006	0	%36.84
Not ESG 1+ESG 2+ESG 5+ESG 6	0.994	31	0.0006	0	%36.84
Not ESG 1+ESG 2+ESG 5+ESG 6 +ESG 4 (SLC)	0.985	31.2	0.0006	%0.65	%36.84
Not ESG 1+ESG 2+ESG 5+ESG 6 +ESG 4+ESG 7	0.9390	75.07	0.00026	%142.16	%72.63
No ESG (equals LC-Testing with length 2)	0.9353	84.65	0.00025	%173.0645	%73.68
LC Testing (length 2+3+4)	0.9999	31.00	0.00095		

ESGs are considered for calculating R_c. Then, data related to ESGs is removed step by step from R_c calculation. R_c is re-calculated at every turn and compared with LC testing. After removing them, the changes become apparent. Finally, the percentages of changes in the parameters are compared to LC testing, which are calculated as described in Eq. (8).

$$CP = \frac{|p^k - p^{LC}|}{|p^{LC}|} * 100 \tag{8}$$

where p^k shows the values of model parameters obtained after removing k-th component and p^{LC} shows the parameters of LC testing.

Table 11 shows the new combined reliabilities and changes in model parameters *CP*, indicating that removing ESG 7 has caused sudden decrease in R_c. When looking at Table 11 from the bottom-up, it can be seen that R_c has increased from 0.9353 to 0.985 as a result of SLC testing with length 3 for ESG 3 and ESG 7. In addition, maximum changes in "a" (representing the expected number of faults to be detected) and "b" (representing the fault detection rate) occur when removing ESG 7 and ESG 3.

Table 12 Effort comparison of LC and SLC (length 3 testing).

length 3	FR (length 3)	LC (length 3)	SLC (length 3; ESG 3+7)	saving SLC vs. LC	saving SLC vs. FR
# events (CES)	1589	1430	1012	29%	36%
# events (FCES)	73053	15150	10634	30%	85%
Total	74642	16580	11646	30%	84%

5.4 Results of the Case Study

When performing SLC testing with length 3 for only ESG 3 and ESG 7, SLC reached a reliability level close to the ones achieved by LC and FR with one difference: the test effort could be reduced even further by approximately 30% with length 3 testing for SLC when directly compared to length 3 testing for LC (see **Table 12**). Compared to FR testing, the test effort has even been reduced by 84%.

5.5 Threats to Validity

Layer-centric (LC) testing has been introduced to reduce the costs of test case generation and test execution for large systems that are modeled in several hierarchical layers. The novel, selective LC strategy, introduced in this paper, brings further valuable control and cost reduction capabilities for the test process. The results of the empirical study have been far above the expectations. However, there are also some limitations and threats to validity which should be mentioned.

Test Approach

The focus of the approach is on testing based on ESGs for detecting faults in sequences of events. However, the real SUC is usually more complex than its model; some aspects might be neglected. Therefore, in reality events may have complex side-effects, and the test data thus generated based on the model might not consider these side effects. Moreover, test data influence test sequences, and thus events may have further side-effects, and the test data thus generated might not be the most appropriate for testing the system. To reduce this inaccuracy, we extended ESG by decision tables [48]. Nevertheless, the influence of test data is still needed to be eliminated to compare the true fault detection between the FR, LC and SLC approach. Thus, the approach creates test data assuming that they have no influence on the detected faults. This can be seen as oversimplifying and thus restrictive; however, the number and severity of the faults detected encourage the robustness of the approach (see Refs. [3], [43] for further examples of the fault detection effectiveness of ESG approach). Moreover, ESG concentrates on detection of the faults on events and on their order. Other types of faults might be detected by chance.

Mainly one large component of a large, commercial system is experimentally tested. However, brief experiments with other components encourage anticipation of similar effects of the test suite reduction. The chosen sub-system considered here is independent of the others so that it can be viewed as a system on its own. Thus, SUC is an impartially-chosen system. However, these results heavily depend on the given SUC and their transferability to other systems requires further empirical studies.

Reliability Determination Approach

Most of reliability growth models, which are used also in this paper, assume that fault correction introduces no new faults (perfect debugging). This assumption is often criticized for not being realistic. One can, however, very easily find out whether the model chosen is appropriate or not, because the model will be applied not only once, but subsequently many times to SUC, as common in statistics. The predicted reliability values will be compared with the measured, real ones, again and again. In case the model selected is not appropriate, the correlation factor will be unacceptable low. Moreover, if the fault correction is not perfect, the next test will reveal this flaw, and the reliability will not grow, which in turn, will indicate that the model selected is not the proper one. In our experiments, a good correlation could be achieved, and reliability growth could be observed.

Reliability analysis is useful in determining when to stop testing. The reliability estimations presented in Section 5 clearly demonstrate the applicability of those models to our empirical study. However, since reliability estimations heavily depend on the given fault data, there is transferability concern for applicability of reliability growth models used in our empirical study.

6. Conclusion and Future Work

Model-based testing is an attractive approach for testing since, depending on the underlying model features and the test criterion considered, test cases can be derived systematically, even automatically [2]. This improvement is demonstrated by an empirical study presented in Section 5.

For the SUC used in the empirical study, LC testing reduced the test effort by approximately 80% with respect to full-resolution (FR) testing. It turned out that the testing effort could be further reduced by approximately 30% using SLC testing compared to LC testing. When compared to the FR testing, SLC testing reduced the test suites by approximately 85% at a fault detection rate of 80%. This fact encourages performing SLC testing over LC testing.

As in our previous work [2], where we compared the fault detection ability of LC testing over FR testing using a reliability theoretical analysis, we repeated the same method for the fault detection ability of the test suites generated by SLC testing to compare with the fault detection ability of the test suites generated by the FR testing by means of a reliability theoretical analysis. This analysis shows that SLC testing achieves a reliability level close to the one achieved by FR testing.

To sum up, the empirical evaluation, conducted using a commercial web portal, shows that SLC testing can reduce the test effort over the LC approach, which, in turn, can significantly re-

duce the effort over the baseline FR testing approach. More importantly, both the reductions have only a small negative impact on fault detection capability.

Future work remains for checking and transferring the results achieved in the empirical study to other systems or models, for example, for testing web service compositions, since the results heavily depend on the given SUC, its development process, and on many more aspects. Moreover, soft computing techniques can be used to consider almost all types of fault data, which in turn, can help to overcome the burden of data dependency problem for reliability determination.

References

[1] Belli, F. and Budnik, C.J.: Test minimization for human-computer interaction, *Journal of Applied Intelligence*, Vol.26, No.2, pp.161–174, Springer (2007).

[2] Belli, F., Güler, N. and Linschulte, M.: Does Depth Really Matter? On the Role of Model Refinement for Testing and Reliability, *IEEE 35th Annual Computer Software and Applications Conference, COMPSAC 2011*, pp.630–639 (2011).

[3] Belli, F., Güler, N. and Linschulte, M.: Are longer test sequences always better? A Reliability Theoretical Analysis, *4th IEEE International Conference on Secure Software Integration and Reliability Improvement, SSIRI 2010*, pp.78–85 (2010).

[4] Mathur, A.P.: *Foundations of software testing*, Addison-Wesley Longman (2008).

[5] Binder, R.V.: *Testing object-oriented systems: models, patterns, and tools*, Addison-Wesley Longman Publishing Co., Inc. (2008).

[6] Myers, G.J.: *Software Reliability: Principles and Practices*, John Wiley & Sons, Inc. (1976).

[7] Myers, G.J.: The art of software testing, John Wiley & Sons, Inc. (1979).

[8] Beizer, B.: *Software testing techniques*, 2nd ed., Van Nostrand Reinhold Co. (1990).

[9] Utting, M. and Legeard, B.: *Practical model-based testing: A tools approach*, Morgan Kaufmann Publishers Inc. (2006).

[10] Grabowski, J., Hogrefe, D., Réthy, G., Schieferdecker, I., Wiles, A. and Willcock, C.: An introduction to the testing and test control notation (TTCN-3), *Computer Networks: The International Journal of Computer and Telecommunications Networking - ITU-T system design languages* (*SDL*), Vol.42, No.3, pp.375–403 (June 2003).

[11] Ammann, P. and Offutt, J.: Introduction to Software Testing, Cambridge University Press (2008).

[12] Bochmann, G.V. and Petrenko, A.: Protocol testing: Review of methods and relevance for software testing, *Proc. 1994 International Symposium on Software Testing and Analysis* (*ISSTA*), pp.109–124, ACM (1994).

[13] Sabnani, K. and Dahbura, A.: A protocol test generation procedure, Computer Networks and ISDN Systems, Vol.15, No.4, pp.285–297 (1988).

[14] Parnas, D.L.: On the use of transition diagrams in the design of a user interface for an interactive computer system, *Proc. 24th National Conference*, pp.379–385, ACM (1969).

[15] Shehady, R.K. and Siewiorek, D.P.: A method to automate user interface testing using variable finite state machines, *Proc. 27th International Symposium on Fault-Tolerant Computing* (*FTCS*), pp.80–88, IEEE Computer Society (1997).

[16] White, L. and Almezen, H.: Generating test cases for GUI responsibilities using complete interaction sequence, *Proc. 11th International Symposium on Software Reliability Engineering* (*ISSRE*), pp.110–121 (2000).

[17] Chow, T.S.: Testing software design modeled by finite-state machines, *IEEE Trans. Software Engineering*, Vol.SE-4, No.3, pp.178–187 (1978).

[18] Memon, A.M., Soffa, M.L. and Pollack, M.E.: Coverage Criteria for GUI Testing, *8th European Software Engineering Conference Held Jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ESEC/FSE-9, pp.256–267, ACM (2001).

[19] Belli, F.: Finite-State Testing and Analysis of Graphical User Interfaces, *12th International Symposium on Software Reliability Engineering, ISSRE 2001*, pp.34–43 (2001).

[20] Belli, F., Beyazit, M. and Memon, A.: Testing is an Event-Centric Activity, *The 6th International Conference on Software Security and Reliability* (*SERE*) (2012).

[21] Gaudel, M.-C.: Testing can be formal, too, *Proc. 6th International Joint Conference CAAP/FASE on Theory and Practice of Software Development* (*TAPSOFT*), pp.82–96, Springer-Verlag (1995).

[22] Koufareva, N.I., Petrenko, A. and Yevtushenko, N.I.: Test generation driven by user-defined fault models, *Proc. 12th International Workshop on Testing Communicating Systems: Method and Applications* (*IWTCS*), pp.215–236 (1999).

[23] Fujiwara, S.V., Bochmann, G., Khendek, F., Amalou, M. and Ghedamsi, A.: Test selection based on finite state models, IEEE Trans. Software Engineering, Vol.17, No.6, pp.591–603 (1991).

[24] Zhu, H., Hall, P.A.V. and May, J.H.R.: Software unit test coverage and adequacy, *ACM Computing Surveys*, Vol.29, No.4, pp.366–427 (1997).

[25] Arcuri, A.: A Theoretical and Empirical Analysis of the Role of Test Sequence Length in Software Testing for Structural Coverage, *IEEE Trans. Software Engineering*, Vol.38, pp.497–519 (2012).

[26] Jourdan, G.V., Ural, H., Yenigün, H. and Zhang, J.: Lower bounds on lengths of checking sequences, *Formal Aspects of Computing*, Vol.22, No.6, pp.667–679 (2010).

[27] Aho, A.H., Dahbura, A.T., Lee, D. and Uyar, M.: An optimization technique for protocol conformance test generation based on uio sequences and rural chinese postman tours, *IEEE Trans. Communications*, Vol.39, No.11, pp.1604–1615 (1991).

[28] Thimbleby, H.: The directed Chinese Postman Problem, *Journal of Software - Practice and Experience*, Vol.33, No.11, pp.1081–1096 (2003).

[29] Burkard, R., Dell'Amico, M. and Martello, M.: Assignment Problems, *Society for Industrial and Applied Mathematics*, Philadelphia (2009).

[30] Suhl, L. and Mellouli, L.: *Optimierungssysteme: Modelle, Verfahren, Software, Anwendungen*, Springer Berlin Heidelberg (2005).

[31] Parnas, D.L.: On the Criteria To Be Used in Decomposing Systems into Modules, *Comm. ACM*, Vol.15, No.12, pp.1053–1058 (1972).

[32] Memon, A.M., Pollack, M.E. and Soffa, M.L.: Hierarchical GUI Test Case Generation Using Automated Planning, *IEEE Trans. Software Engineering*, Vol.27, No.2, pp.144–155 (2001).

[33] Paiva, A.C.R., Tillmann, N., Faria, J.C.P. and Vidal, R.F.A.M.: Modeling and Testing Hierarchical GUIs, *Proc. 12th International Workshop on Abstract State Machines*, pp.8–11 (2005).

[34] Andrews, A.A., Offutt, J. and Alexander, R.T.: Testing Web applications by modeling with FSMs, *Software and Systems Modeling*, Vol.4, No.3, pp.326–345 (2005).

[35] Reza, H., Endapally, S. and Grant, E.: A Model-Based Approach for Testing GUI Using Hierarchical Predicate Transition Nets, *International Conference on Information Technology* (*ITNG'07*), pp.366–370 (2007).

[36] Lyu, M.R.: *Handbook of Software Reliability Engineering*, McGraw-Hill (1996).

[37] IEEE STD, Recommended Practice on Software Reliability, 1633-2008, c1-72 (2008).

[38] IEC 62628 ed1.0, Guidance on software aspects of dependability (2012).

[39] AIAA STD: Recommended Practice for Software Reliability, AIAA R-013-1992 (1992).

[40] Gökhale, S. and Trivedi, K.: Analytical Models for Architecture-Based Software Reliability Prediction: A Unification Framework, *IEEE Trans. Reliability*, Vol.55, No.4, pp.578–590 (2006).

[41] Yacoub, S., Cukic, B. and Ammar, H.H.: A scenario-based reliability analysis approach for component-based software, *IEEE Trans. Reliability*, Vol.53, No.4, pp.465–480 (2004).

[42] Krishnamurthy, S. and Mathur, A.P.: On the estimation of reliability of a software system using reliabilities of its components, *Proc. 8th International Symposium on Software Reliability Engineering*, pp.146–155 (1997).

[43] Belli, F., Budnik, C.J. and White, L., Event-based modeling, analysis and testing of user interactions: Approach and case study, *Journal of Software Testing, Verification and Reliability* (*STVR*), Vol.16, No.1, pp.3–32, John Wiley & Sons, Ltd. (2006).

[44] West, D.B.: *Introduction to Graph Theory*, Prentice Hall (1996).

[45] Dolbec J. and T. Shepard: A Component Based Software Reliability model, *Proc. 1995 Conference of the Centre for Advanced Studies on Collaborative Research*, p.19 (1995).

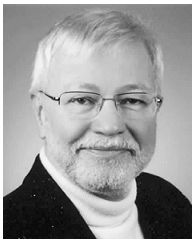
[46] Birolini, A.: *Reliability Engineering: Theory and Practice*, Springer (2007).

[47] Shibata, K., Rinsaka, K. and Dohi, T.: Metrics-Based Software Reliability Models Using Non-homogeneous Poisson Processes, *Proc. 17th International Symposium on Software Reliability Engineering*, pp.52–61, IEEE Computer Society (2006).

[48] Belli, F. and Linschulte, M.: On Negative Tests of Web Applications, *Annals of Mathematics, Computing & Teleinformatics*, Vol.1, No.5, pp.44–56 (2007).

[49] Williams T.W. and Parker, K.P.: Design for Testability - A Survey,

IEEE Trans. Comp., Vol.31, pp.2–15 (1982).
[50] Fraser, G. and Arcuri, A.: Handling test length bloat, Software Testing, *Verification and Reliability*, Vol.23, No.7, pp.553–582 (2013).
[51] Offutt, A.J., Rothermel, G., Christian Zapf: An Experimental Evaluation of Selective Mutation, *Proc. 15th International Conference on Software Engineering* (*ICSE '93*), pp.100–107 (1993).
[52] Chen, Y., Rosenblum, D.S. and Vo, K.P.: TestTube: A system for selective regression testing, ICSE '94 *Proc. 16th International Conference on Software Engineering*, pp.211–220 (1994).
[53] Hirayama, M., Yamamoto, T., Okayasu, J., Mizuno, O. and Kikuno, T.: A selective software testing method based on priorities assigned to functional modules, *Proc. 2nd Asia-Pacific Conference on Quality Software*, pp.259–267 (2001).
[54] Hirayama, M., Mizuno, O. and Kikuno, T.: Test item prioritizing metrics for selective software testing, *IEICE Trans. on Information and Systems*, Vol.87, No.12, pp.2733–2743 (2004).
[55] Steinert, B., Haupt, M., Krahn, R. and Hirschfeld, R.: Continuous Selective Testing, *Proc. 11th International Conference on Agile Processes in Software Engineering and Extreme Programming* (*XP 2010*), pp.132–146 (2010).
[56] Andrews, A. and Do, H.: Trade-Off Analysis for Selective versus Brute-Force Regression Testing in FSMWeb, *Proc. 15th IEEE International Symposium on High-Assurance Systems Engineering*, pp.184–192 (2014).
[57] Tyagi K. and Sharma, A.: A rule-based approach for estimating the reliability of component-based systems, *Advances in Engineering Software*, Vol.54, pp.24–29 (2012).
[58] Singh, L.K., Vinod, G. and Tripathi, A.K.: Approach for parameter estimation in Markov model of software reliability for early prediction: a case study, *IET Softw.*, Vol.9, No.3, pp.65–75 (2015).
[59] Li, K., Liu, L., Zhai, J., Kosgoftaar, T.M., Shaoa, M. and Liua, W.: Reliability Evaluation Model of Component-Based Software Based on Complex Network Theory, *Qual. Reliab. Engng. Int.*, (wileyonlinelibrary.com), DOI: 10.1002/qre.2033 (2016).
[60] Goel, A.L. and Okumoto, K.: Time-dependent error-detection rate model for software reliability and other performance measures, *IEEE Trans. Reliability*, Vol.R-28, No.3, pp.206–211 (Aug. 1979).
[61] Goel, A.: Software reliability models: Assumptions, limitations, and applicability, *IEEE Trans. Software Engineering*, Vol.SE-11, No.12, pp.1411–1423 (Dec. 1985).
[62] Yamada, S., Ohba, M. and Osaki, S.: S-shaped reliability growth modeling for software error detection, *IEEE Trans. Reliability*, Vol.R-32, No.5, pp.475–484 (Dec. 1983).
[63] Zhao, M. and Xie, M.: On the log-power NHPP software reliability model, *Proc. 3rd IEEE International Symposium on Software Reliability Engineering* (*ISSRE*), pp.14–22, IEEE Computer Society (1992).
[64] Duane, J.T.: Learning curve approach to reliability monitoring, *IEEE Trans. Aerospace*, Vol.2, No.2, pp.563–566 (Apr. 1964).
[65] Musa, J.D. and Okumoto, K.: A logarithmic poisson execution time model for software reliability measurement, *Proc. 7th International Conference on Software Engineering* (*ICSE*), pp.230–238, IEEE Computer Society (1984).



Fevzi Belli completed his Ph.D. in formal methods for self-correction features in sequential systems in 1978 and his "Habilitation" (German Post-Doctoral degree) in software engineering in 1986 at Berlin Technical University. In 1983, he was awarded a professorship at the University of Applied Sciences in Bremerhaven; in 1989 he moved to the University of Paderborn. Between 2014–2016, he has been a full professor at Izmir Institute of Technology. Currently, he is emiritus at University of Paderborn.

Nevin Güler Dincer received her Ph.D. degree in Mathematics from Mugla SıtkıKoçman University, Turkey. She is currently working as an assistant professor with department of statistics at the same university. She is mainly working on fuzzy modeling, fuzzy time series, fuzzy clustering and soft computing techniques.

Michael Linschulte studied Business Computing Systems and received his Ph.D. degree at University of Paderborn before he joined Adagon in Cologne. His research interests include analysis and testing of web applications as well as web services by means of model-based testing. Further, he is interested in test automation, software reliability and fault tolerance.

Tugkan Tuglular received his 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 Izmir Institute of Technology since 2000. He is interested in model-based testing and test automation.