This article has been accepted for publication in a future issue of a journal/proceedings, but has not been fully edited. Content may change prior to final publication. Citation information: Belli, F., Güler, N., Linschulte, M., Does 'Depth' Really Matter? On the Role of Model Refinement for Testing and Reliability, Proc. 35th Annual IEEE Computer Software and Applications Conference (COMPSAC), Best Paper, pp.630-639

Does "Depth" Really Matter? On the Role of Model Refinement For Testing and Reliability

Fevzi Belli[1], Nevin Güler[2], Michael Linschulte[1]
[1] Electrical Engineering and Mathematics, University of Paderborn, Paderborn, Germany
e-mail:{belli,linschu}@upb.de
[2] Departments of Statistics, University of Mugla, Mugla, Turkey
e-mail: nguler@mu.edu.tr

Abstract—**Model-based testing attempts to generate test cases from a model focusing on relevant aspects of a given system under consideration (SUC). When SUC becomes too large to be modeled in a single step, existing design techniques usually require a modularization of the modeling process. Thereby, the refinement process results in a decomposition of the model into several hierarchical layers. Conventional testing requires the refined components be completely replaced by these subcomponents for test case generation. Mostly, this resolution of components leads to an oversized, large model where test case generation becomes very costly, and the generated test case set is very large leading to infeasible long test execution time. To solve these problems, we present a new strategy to reduce (i) the number of test cases, and (ii) the costs of test case generation and test execution. For determining the trade-off due to this cost reduction, the reliability achieved by the new approach is compared with the reliability of the conventional approach. A case study based on a large web-based commercial system validates the approach and discusses its characteristics. We found out that the new approach could detect about 80% of the faults for about 20% of the test effort compared with the conventional approach.**

Keywords-model-based testing; model refinement; event sequence graphs; software reliability; assignment problem.

I. INTRODUCTION

Software testing aims to increase confidence in the software by checking whether the software does what is expected to do or not. The reason for an error might be an incorrect step, process or data definition in a computer program, also called as *fault;* an incorrect result as the result of a fault is called a *failure* [1].

In recent years, many approaches have been developed that are addicted to software testing. Model-based testing (MBT) creates a model of the system's behavior out of the specification. Most of these models are graph-based, as collected in UML [2]. MBT enables testers to focus on the relevant aspects of the system under consideration (SUC) and, depending on the underlying model features and the test criterion considered, the ability of automatic generation of test cases (forming *test suites*) even if the source code of SUC is not available as well as update and reuse of these test cases with evolving requirements.

A common problem with deriving tests from the model is that the complexity of SUC can cause a state space explosion. Therefore, most of the existing techniques require the refinement of the starting model by additional models resulting in a hierarchical decomposition of the SUC. An example is given in Fig. 1 where vertex *b* of the model is refined by an additional model. For testing, refined components have to be resolved completely before test sequences are generated, leading to a *fully resolved (FR)* model (Fig. 2).

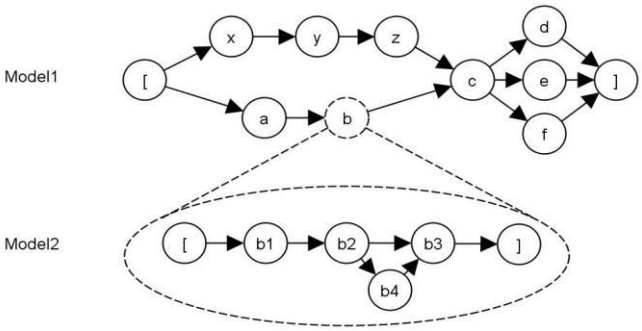

Figure 1. A compound vertex *b* of the model is refined.

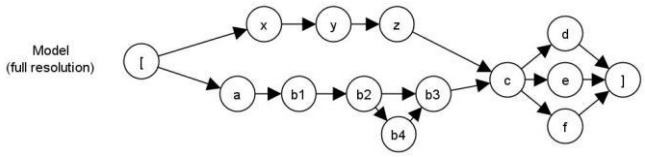

Figure 2. Completed (full resolved) version of the model given in Fig. 1.

Very often, a full resolution of refined vertices results in a large model. Assuming that the run-time complexity of finding a minimal test set is $O(|V|^3)$ [3], where |V| denotes the number of vertices, it is quite obvious that the test generation effort increases with the increasing model size and number of hierarchy layers.

Our previous work [4] analyzed the effect of test sequence length on the failure detection capability of model-based testing for a single layer model. However, the impact of model refinement has not been considered. The present paper presents a *layer-centric* (*LC*) test generation method to fill this gap and answering following questions:

- *How can the test costs be reduced if the modeling process produces a hierarchical set of models?*
- *What is the trade-off, i.e., the impact of this cost reduction on the overall system reliability?*

The first question requires a new strategy for test generation and minimization. The second question requires the comparison of the reliability level achieved following the

new LC strategy with the one achieved by following the conventional strategy, using the fully resolved model.

The next section summarizes related work and discusses the impact on our work. Section III introduces the terminology and notion used in our approach and describes the new test case generation strategy. Section IV reviews existing reliability models to select the ones that fit best for the comparison of LC and FR strategy. Section V validates the approach and determines its characteristic features over a case study based on a commercial web-based software system. Section VI concludes the paper summarizing the results and outlines our research work planned.

II. Related Work

A broad variety of formal or informal models exist for modeling and testing software as recommended in standards such as UML [2]. Depending on user needs, those models describe SUC at different levels of granularity and preciseness. Graph-based models consist of nodes and arcs that connect the nodes. The semantics associated with these nodes and arcs determine the level of granularity of SUC description. Event sequence graphs (ESG) [5] can be used for modeling, analysis and validation of system behavior and user interface requirements prior to implementation and testing of the code. The present paper chooses ESG notation because it intensively uses formal notions and algorithms known from graph theory and automata theory, which are relevant to the approach introduced in this paper.

Memon et al. [6] use an automatic planning system to generate test cases from GUI events and their interactions called Planning Assisted Tester for graphical systems. Paiva et al. [7] presented an approach based on Hierarchical Finite State Machines where the hierarchical structure finds special attention during the test case generation process. Andrews et al. [8] proposed a system-level testing technique that combines test generation based on FSMs with constraints. Reza et al. [9] use a high level Petri net known as Hierarchical Predicate Transition Net to model the behavior of SUC and to generate adequate test cases using this model.

All of the mentioned approaches engage hierarchical structures and focus mainly on GUI testing since ESGs are user-centric and therefore are ideally suited to system/GUI-based testing. However, there is no approach that makes use of the hierarchy for producing optimized test suites.

Observations obtained from software testing can be used to determine the software reliability (SR) [10], [11] by means of software reliability models. Several software reliability models have been developed under different sets of assumptions during the last three decades [12]. Non-homogenous Poisson Process (NHPP) models are a widely used class among existing software reliability models because of their simplicity and compatibility. NHPP models assume that the failure data follow Poisson distribution with a mean parameter. These models differ in their mean value function and failure intensity function. Musa-Okumoto (M-O), Goel-Okumoto (G-O) and Delayed S-Shaped (D-S) [12] are well-known members of this class of models.

SR can also be used to compare the fault detection ability of different software testing techniques and to determine when to stop testing. In our previous work [4], we examined the role of test sequence length with respect to its failure detection effectiveness. Consequently, the approach used in this study can also be applied to decide whether to stop testing or to continue testing by increasing the sequence length for detection of additional failures. Similarly, Arcuri [13] investigated the role that the length of test sequences plays in software testing, in particular, branch coverage, and has empirically shown that longer test sequences can improve the results. Contrarily, our work [4] showed that the number of failures detected decreases strictly for raising length of test sequences since

- most of the failures can be detected by event sequences of minimal length, that is, 2 and
- no new failures are likely to be detected by longer event sequences, that is, of length 4 and 5.

To our knowledge, there is no research work comparing the reliability levels achieved by LC and FR.

III. Test Generation and Minimization

This Section summarizes the technique used for graph-based modeling of SUC and introduces the new strategy to generate and select test cases for constructing an optimal test suite.

A. Event Sequence Graphs

Belli et al. [5] introduced an event-based approach called *event sequence graph* (ESG) for testing interactive systems. Vertices of the ESG represent externally observable phenomena ("events"), that is, an environmental or a user stimulus, or a system response, punctuating different stages of system activity. Directed edges connecting two events define allowed sequences among these events.

Definition 1. An $ESG = (V,E)$ is a directed graph where V is a finite set of vertices uniquely labeled by some input symbols of the alphabet Σ, denoting events, and E: $V \rightarrow V$, a precedence relation, possibly empty, on V. The elements of E represent directed edges between the vertices in V. ■

The semantics of an ESG is as follows. Given two vertices a and b in V, a directed edge ab from a to b indicates that event b follows event a, defining an *event pair* (*EP*) ab. The remaining pairs $\bar{E}$ that can be constructed by all combinations $\hat{E}=V\times V$ of the nodes given in the alphabet Σ, but not in the ESG, that is, $\bar{E}= \hat{E}\backslash E$, form the set of *faulty event pairs* (*FEP*). The set of FEPs constitutes the *complement* of the given ESG, which is symbolized as $\overline{ESG}$. As a convention, a dedicated start vertex, e.g., I, remarks the *entry* vertices Ξ of the ESG whereas a final vertex, e.g., J, represents the *exit* vertices Γ. Note that I and J are not included in Σ.

A vertex representing a single, self-contained event is called *atomic* event/vertex. Besides, vertices can be refined by another ESG (see Fig. 1) resulting in a hierarchy of models, i.e., they are *compound* events/vertices consisting of atomic events and/or even other compound events.

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 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 ESG given. Accordingly, a *faulty event sequence* (*FES*) of length *n* consists of *n-2* concluding, subsequent EPs and ends with an FEP. An FES is *complete* if it starts at the initial vertex of the ESG; in this case, it is called *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 start at the entry and end at the first node of the considered FES.

CESs and FCESs form test cases to 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). 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). If this is feasible, the corresponding FCES is marked as failed. However, before test sequences can be generated, compound vertices are to be resolved according to Definition 2.

Definition 2. Given an ESG, say $ESG_1 = (V_1, E_1)$, a vertex $v \in V_1$, and an ESG, say $ESG_2 = (V_2, E_2)$, then replacing v by ESG_2 produces a *refinement* of ESG_1, say $ESG_3 = (V_3, E_3)$ with $V_3 = V_1 \cup V_2 \setminus \{v\}$, and $E_3 = E_1 \cup E_2 \cup E_{pre} \cup E_{post} \setminus E_{1replaced}$ (\ is the set difference equation), wherein $E_{pre} = N^-(v) \times \Xi(ESG_2)$ (connections of the predecessors of v with the entry nodes of ESG_2), $E_{post} = \Gamma(ESG_2) \times N^+(v)$ (connections of exit nodes of ESG_2 with the successors of v), and $E_{1replaced} = \{(v_i, v), (v, v_k)\}$ with $v_i \in N^-(v)$ and $v_k \in N^+(v)$ (replaced edges of ESG_1). ■

Note that $N^-(v)$ remarks the predecessors of a vertex v and $N^+(v)$ the successors of v.

Example 1. In Fig. 1, the refinement of vertex *b* of Model1 is given as Model2. The Model given in Fig. 2 is the resolved version of Model1. More precisely, Model1 is given as V_1={a, b, c, d, e, f, x, y, z}, E_1={(a,b), (b,c), (c,d), (c,e), (c,f), (x,y), (y,z), (z,c)}, Ξ(Model1)={a, x} and Γ(Model1)={d, e, f}. In the refinement, i.e., Model2 of the compound event *b*, the predecessors and successors are $N^-(b)$={a}, $N^+(b)$={c} and the refinement of Model2 is given by V_2={b1, b2, b3, b4}, E_2={(b1, b2), (b2,b3), (b2,b4), (b4, b3)}, Ξ(Model2)={b1} and Γ(Model2)={b3}. The resulting (full resolved) model shown in Fig. 2 is represented by

Model3=(V_3,E_3) with

$V_3 = V_1 \cup V_2 \setminus \{b\}$ = {a, b, c, d, e, f, x, y, z} ∪ {b1, b2, b3, b4}\{*b*} = { a, b1, b2, b3, b4, c, d, e, f, x, y, z } and

$E_3 = E_1 \cup E_2 \cup E_{pre} \cup E_{post} \setminus E_{1\ replaced}$ = {(a,b), (b,c), (c,d), (c,e), (c,f), (x,y), (y,z), (z,c)} ∪ {(b1, b2), (b2,b3), (b2,b4), (b4, b3)} ∪ {(a,b1)} ∪ {(b3, c)}\{(a,b), (b,c)}= {(c,d), (c,e), (c,f), (x,y), (y,z), (z,c), (b1, b2), (b2,b3), (b2,b4), (b4, b3), (a,b1), (b3, c)}

Ξ(Model3)={a, x} and

Γ(Model3)={d, e, f} ■

B. Test Case Generation

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* (*CPP*) that attempts to find the shortest path or cycle in a graph by visiting each arc [3].

As already mentioned before, hierarchical models are supposed 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 [3]. The number of FCESs for negative testing increases with increasing number of vertices since $|FCES| = |V|^2 - |E|$.

Example 2: The strategy explained in [3], [5] delivers for the Model given in Fig. 2 following CESs as test sequences:

CES1=[x y z c d]
CES2=[a b1 b2 b3 c e]
CES3=[a b1 b2 b4 b3 c f]

Following, we present some of the resulting FCESs as negative test cases.

FCES1=[x a
FCES2=[x b1

Note that the set of negative tests has $|FCES| = |V|^2 - |E| = 12^2 - 12 = 132$ elements; adding the three positive test cases above results to a total of 135 test cases for this example. ■

For enabling the solution of the CPP, the given graph is to be extended by additional edges until it forms an *Eulerian graph.* An Eulerian graph has a cycle traversing each edge exactly once, and which returns to the start vertex. A directed graph is Eulerian if it is strongly connected and each of its vertices $v \in V$ has equal indegree and outdegree that are defined as follows.

Definition 3. The number of edges going into a vertex v is the *indegree* written $\delta^-(v)$, and the number of edges pointing out of a vertex v is the *outdegree* written $\delta^+(v)$. Let δ be the difference between the in- and outdegrees: $\delta(v) = \delta^-(v) - \delta^+(v)$. If $\delta(v) = 0$, vertex v is called *balanced.* ■

Following Definition 3 leads to the conclusion that a directed graph is Eulerian if every vertex is balanced, that is, $\delta(v)=0 \ \forall \ v \in V$, respectively. The resulting *Eulerian cycle*, which can be obtained by a standard algorithm in O(|V|*|E|) time [14], is a minimal solution to the CPP if the set of added edges is minimal. For determining the set of minimal edges, two sets *A* and *B* have to be generated with

$$A = \{v_i \mid i \in \{1,\ldots,\delta(v)\} \wedge \delta(v) > 0\}$$
$$B = \{v_i \mid i \in \{1,\ldots,-\delta(v)\} \wedge \delta(v) < 0\}$$

Fig. 3 shows the degrees for each vertex of Fig. 2 with set *A*={*],],b3*} and set *B*={*[,b2,c*}. Note that an edge is added from end vertex *]* to start vertex *[* in Fig. 3 to fulfill the entity of strong connectivity.

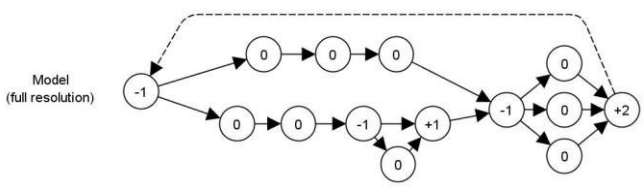

Figure 3. Degrees of the vertices of the ESG given in Fig. 2.

The challenge is to assign each element of set *A* to exactly one element of set *B* so that there is no element in both sets which is unassigned and there is no other assignment whose number of edges to be added (according to the assignment) is lower. This leads to the *assignment problem (AP)* [15] which attempts to answer the question of how to assign *n* items (agents) to *n* other items (tasks), incurring some *cost* that may vary depending on the agent-task assignment. It is required to perform all tasks by assigning exactly one agent to each task in such a way that the *total cost* of the assignment is minimal. Formally, an assignment problem minimizes the o*bjective function* (1) for a given $n \times n$ cost matrix c_{ij} which fulfills the given constraints (2), (3) and (4) at the same time.

$$\min \quad \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij} x_{ij} \tag{1}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} x_{ij} = 1 \quad (i = 1,...,n) \tag{2}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \quad (j = 1,...,n) \tag{3}$$

$$x_{ij} \in \{0,1\} \quad (i, j = 1,...,n) \tag{4}$$

In our case, c_{ij} defines the number of edges of the shortest path (as costs) between vertex $i \in A$ and vertex $j \in B$ and n the number of elements of set A or B, respectively. After minimization, $x_{ij}=1$ 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, $n=|A|=|B|$.

Example 3: Table I 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 I, that is, $x_{ij}=1$ for the dark grey boxes. Furthermore, there should be no other assignment the sum of costs of which is lower. According to Table I, following shortest paths have to be added to the ESG given in Fig. 2 to create a minimal Eulerian cycle: *]→[,]→b2, b3→c.* ■

TABLE I. THE RESULTING COST AND X_{ij} MATRIX OUT OF FIG. 3

c_{ij}	[	b2	c
]	1	4	5
]	1	4	5
b3	4	7	1

x_{ij}	[	b2	c	∑
]	1	0	0	1
]	0	1	0	1
b3	0	0	1	1
∑	1	1	1	

It is known from graph theory [14] that the construction of a minimal set of edges, which creates an Eulerian graph, leads to the assignment problem that can be solved in different ways:

i) *Complete enumeration method:* All combinations of assignments are prepared and an assignment involving the minimum cost is selected. This method might be suitable for assignment problems of small size but it is not suitable for real world applications since the number of possible combinations is *n!*, e.g., an assignment matrix with 10 rows/columns would have round about three and a half million of combinations to evaluate.

ii) *Simplex method:* APs can be formulated as a linear programming model. The simplex method is a well-known algorithm for solving linear programming problems. Although the simplex method has better runtime complexity than the complete enumeration method, solving APs using the simplex method can also be very time consuming since it has exponential runtime complexity.

iii) *Hungarian method:* One of the fastest methods for solving APs is the Hungarian method [15] which provides a solution in $O(n^3)$ time [15]. Beside the Hungarian method, other $O(n^3)$ solutions are given by Dinic-Kronrod [15] or Cycle Canceling [16].

C. Layer-Centric Testing

Example 1 is intentionally constructed in such a way that the determination of test cases becomes very simple. In large hierarchical models, the determination of test cases becomes very complex. Therefore, we suggest (instead of resolving compound vertices) to generate test cases for each ESG on its own, i.e., we suggest a *layer-centric* (*LC*) test generation method instead of a *full resolution* (*FR*) test generation method. In other words, we leave the compound vertices unresolved within LC-testing. This will reduce the effort of finding a minimal solution since

$$O(|V_{resolved}|^3) > O(|V_1|^3)+\ldots+ O(|V_k|^3)$$

where *k* is the number of single ESGs forming the hierarchy, that is, k=2 for Fig. 1. This strategy will reduce also the number of negative test cases significantly since FEPs between different ESG layers are not considered.

Example 4. Consider Model1 and Model2 of Fig. 1. The optimization algorithm given in [3], [5] generates for Model1 and Model2 following CESs:

Model1:	Model2:
T1=[x y z c d]	T4=[b1 b2 b3]
T2=[a b c e]	T5=[b1 b2 b4 b3]
T3=[a b c f]	

For positive testing, we have 5 test cases (instead of 3, Example 2). The number of resulting FCESs (negative testing) for Model1 is 72 and for Model2 is this number 12. Compared to Example 1, the total number of test cases has been reduced from 135 (see Example 2) to 89. ■

Analysis of Example 3 reveals following problem: Compound vertices, e.g., *b* in Example 3, 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 weight the compound events based on the number of events they include.

Definition 4. The *weight* of a compound vertex is given by the length of the shortest ES with $\alpha(ES)=\varepsilon$ as the entry and $\omega(ES)=\gamma$ as the exit of this compound vertex. ■

Example 5. The weight of Model2 of Fig. 1 is *3*, because the shortest ES possible is [b1 b2 b3]. ■

Example 6. If we take the weight of the compound event into account, the test set of Example 3 modifies as follows.

Model1:
T1=[x y z c d]
T2=[x y z c e]
T3=[a b c f]

Model2:
T4=[b1 b2 b3]
T5=[b1 b2 b4 b3]

■

Now, another problem has to be considered: How can test sequences be executed that contain compound vertices?

A straightforward strategy is to replace the compound vertices by test case(s) generated from the lower-layer ESG. If also this lower-layer ESG 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 7. In Example 6, replacing *b* in T3 by T4, following test sequences can be constructed:

T1=[x y z c d]
T2=[x y z c e]
T3'= [a b1 b2 b3 c f]
T4=[b1 b2 b4 b3] ■

Note that T1, T2, and T3' can be executed at the top layer, using Model1. T4 is to be executed at the next lower layer, using Model2. This can be seen as inconsequent. A solution is needed in which the minimal coverage of Model1 contains node *b* as many times as test cases exist for the compound vertex *b*. This can be achieved by an extension of the assignment matrix of Model1 by additional columns and rows for each vertex that is needed more often. Table II shows the resulting assignment matrix extended by vertex *b*.

TABLE II. THE EXTENDED ASSIGNMENT MATRIX FOR MODEL1

c_{ij}	[	c	b
]	1	5	5
]	1	5	5
b	3	1	8

Example 8. Taking Table II into account results in the following test sequences for Model1 and Model2:

Model1:
T1=[x y z c d]
T2=[a b c e]
T3=[a b c f]

Model2:
T4=[b1 b2 b3]
T5=[b1 b2 b4 b3]

Replacing vertex *b* in T2 and T3 by T4 and T5, respectively, following test sequences can be constructed for Model1 and Model2:

Combined test case set:

T1=[x y z c d]
T2'=[a b1 b2 b3 c e]
T3'=[a b1 b2 b4 b3 c f] ■

Algorithm 1 (see http://quebec.upb.de/compsac2011.pdf, [17] for a detailed description) describes the new LC testing approach for positive testing in details. Algorithm 1 makes use of Algorithm 2 describing the approach to balance the graph formally. Therefore, Algorithm 2 concretizes the findings above. Algorithm 3 completes the approach by describing how to generate the desired CES on a balanced ESG.

In contrast to generating CESs, calculating FCESs for negative testing is considerably easier. FCESs of ESGs of lower layer are generated first and then returned to the next higher layer where the shortest path [14] 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 4 generates FCESs which makes use of Algorithm 5 to generate FCESs for a single ESG of one layer (see http://quebec.upb.de/compsac2011.pdf, [17] for a detailed description). Whereas Algorithm 4 and Algorithm 5 contain also the way to derive FCESs also for higher length, this will be part of the next Section for CESs.

D. Layer-Centric Testing for Sequence Length > 2

A phenomenon in testing of interactive systems is that failures can be frequently observed but reproduced only in specific context(s). Coverage of event sequences of length >2 can help here. To achieve such coverage(s), the original graph will be transformed to enable the re-use of algorithms for length 2 coverage again. The algorithm for transforming the ESG given is shown in Algorithm 6 (see http://quebec.upb.de/compsac2011.pdf, [17] for a detailed description). Fig. 4 shows the transformation of Model1 of Fig. 1 to achieve a coverage of sequence length 3.

As it can easily be seen in Fig. 4 for (indexed) vertex *c*, a vertex might be contained more than once in the resulting graph. If the transformed graph includes a vertex $v \in V$ of the original graph more than once and if a multiple occurrence of that vertex *v* is needed, it is then not obvious which vertex *v* will minimize the assignment. A solution is to extend the assignment matrix by all occurrences of *v* and redefine the matching constraints.

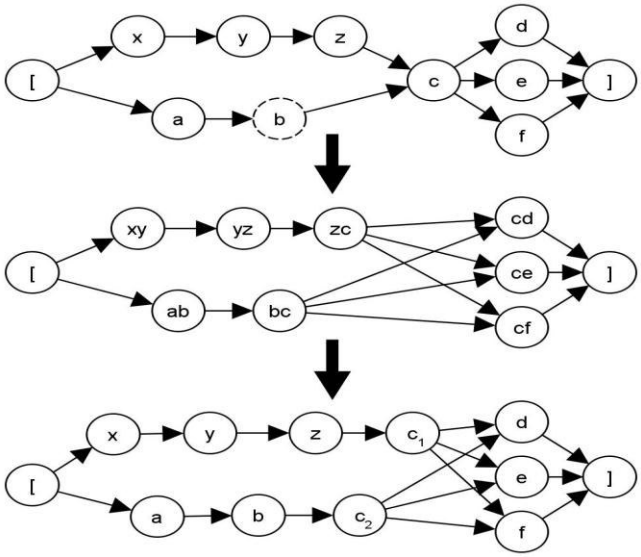

Figure 4. Graph transformation for length 3 coverage

Example 9. Assuming that vertex *c* is needed one more time in our solution, then it is not obvious if vertex c_1 or vertex c_2 of Fig. 4 will minimize the solution. Adding vertex c_1 to the assignment matrix results in costs of 29 (see Table III, left) whereas adding vertex c_2 results in costs of 28 (see Table III, right). That is, vertex c_2 would minimize the solution. ■

TABLE III. THE COST MATRIX FOR FIG. 4 EXTENDED BY c_1 (LEFT) AND c_2 (RIGHT)

c_{ij}	[	c_1	c_1	c_2	c_2	c_1
]	1	5	5	4	4	5
]	1	5	5	4	4	5
d	2	6	6	5	5	6
e	2	6	6	5	5	6
f	2	6	6	5	5	6
c_1	3	7	7	6	6	7

c_{ij}	[	c_1	c_1	c_2	c_2	c_2
]	1	5	5	4	4	4
]	1	5	5	4	4	4
d	2	6	6	5	5	5
e	2	6	6	5	5	5
f	2	6	6	5	5	5
c_2	3	7	7	6	6	6

Considering Example 9 raises the need for a more straightforward approach which solves the given problem within one cost matrix as it can be seen in Table IV. Formally, following set of equations described as a linear program has to be solved in such a case:

$$\min \quad \sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \tag{5}$$

$$\text{s.t.} \quad \sum_{j=1}^{n} x_{ij} = 1 \quad (i = 1,...,k) \tag{6}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \quad (j = 1,...,k) \tag{7}$$

$$\sum_{j=1}^{n}\sum_{i=s(y)}^{e(y)} x_{ij} = 1 \quad (y = 1,...,l) \tag{8}$$

$$\sum_{i=1}^{n}\sum_{j=s(y)}^{e(y)} x_{ij} = 1 \quad (y = 1,...,l) \tag{9}$$

$$\sum_{i=1}^{n}(x_{ij} - x_{ji}) = 0 \quad (j = s(y),...,e(y), y = 1,...,l) \tag{10}$$

$$x_{ij} \in \{0,1\}, \quad (i, j = 1,...,n) \tag{11}$$

Example 10. In Table IV, column/row 6 and 7 have been added and only one of both should be put to the solution w.r.t. minimization purposes. In this case, column/row 7 minimizes the assignment and column/row 6 has to be skipped. According to Table IV, k=5, l=1, $s(1)$=6, $e(1)$=7. Thus, l defines the number of matrix intervals from which only one should be selected (here only one consisting of column/row 6 and 7), $s(y)$ defines the first index of interval $y<=l$ and $e(y)$ defines the last index of interval $y<=l$. k defines the last index of columns/rows which do not belong to an interval. ■

TABLE IV. THE COMBINED COST MATRIX FOR FIG. 4 CONTAINING c_1 AND c_2 SIMULTANIOUSLY

			j						
			1	**2**	**3**	**4**	**5**	**6**	**7**
		c_{ij}	[	c_1	c_1	c_2	c_2	c_1	c_2
i	**1**	]	1	5	5	4	4	5	4
	2	]	1	5	5	4	4	5	4
	3	**d**	2	6	6	5	5	6	5
	4	**e**	2	6	6	5	5	6	5
	5	**f**	2	6	6	5	5	6	5
	6	c_1	3	7	7	6	6	7	∞
	7	c_2	3	7	7	6	6	∞	6

Algorithm 7 (see *http://quebec.upb.de/compsac2011.pdf* [17] for a detailed description) improves Algorithm 2 for balancing a graph to cover sequences of higher length according to the findings described above.

Unfortunately, the given equation system (5)-(11) cannot be resolved by the Hungarian method. An optimal solution (abbreviated as LC_{opt}) can be calculated using linear programming, but it has exponential runtime complexity.

In case that a solution cannot be found in appropriate time, a heuristic approach is given by solving the assignment problem first and then adding the shortest self-cycle for the given vertex v as many times as needed; this strategy is abbreviated as LC_{simple}. LC_{simple} is straightforward and feasible, but not necessarily minimal. See *http://quebec.upb.de/compsac2011.pdf* [17] for a formal description of LC_{simple}.

IV. RELIABILITY ANALYSIS OF THE LC STRATEGY

As demonstrated in the previous sections, LC strategy can dramatically reduce the size of the test suite, and thus the test costs. The critical question is, however, how far this cost reduction will affect the reliability. If the trade-off negatively influences the quality, the saving effect will not persuade to accept the new strategy. Thus, we need a quantitative comparison of the reliability levels achieved by both strategies, LC and FR. This section briefly summarizes and discusses existing software reliability models, which have been used since the early seventies of the last century for reliability determination based on fault data observed during testing of SUC (see also existing standards and guidelines, e.g. [10], [11]). As there is no prior knowledge about the characteristics of the fault data, we have to check several of these models for selecting the one that fits best to the data obtained. This section discusses also the criteria given for the selection.

Starting with the same hierarchical model, test suites are separately generated and executed following LC and FR strategies (Fig. 5). The test results build up the fault data which are to be analyzed to select the best fitting SR model. Finally, the reliabilities of the both strategies, R_{LC} and R_{FR}, will be calculated and compared.

As we want to count each of the faults only once assuming they were corrected upon detection without causing new faults, we will use *Software Reliability Growth Models* (*SRGM*), which assume that the absolute number of faults remaining in the software decreases and thus SR grows.

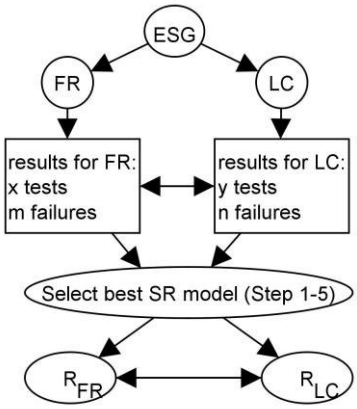

Figure 5. Overview of the reliability analysis

SRGMs can be classified along five different attributes introduced by Musa and Okumoto [12]:

- *Time domain.* Calender time versus execution time,
- *Type.* The distribution of the number of failures experienced by time t is Binomial or Poisson.
- *Category.* The total number of failures that can be experienced in infinite time is considered either as finite or as infinite.
- *Class* (in finite failure category only). Functional form of the failure intensity expressed in terms of time, e.g., exponential, Weibull or gamma class.
- *Family* (in infinite failure category only). Functional form of the failure intensity function expressed in terms of the expected number of failures experienced, e.g., geometric family, power family.

In order to select proper SRGMs for predicting reliability R_{LC} and R_{FR}, following steps will be performed:

➢ *Step 1: 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, number of test cases or execution time. Moreover, there are two types of failure data for SRGMs: time intervals between successive observed failures and the number of failures detected in a specified time interval.

➢ *Step 2: 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 Kolmogorow Smirnov Test (K-S) [18] is one of statistical nonparametric tests used to determine whether a sample (failure data collected) fits the specified distribution. In our case study, we observe that the cumulative number of failures builds up Poisson distribution according to K-S test (in Section V.B, Table VIII).

Poisson type models can be divided into two groups 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 a constant failure intensity. In our case, failure intensity varies with time parameter since faults are only counted once and no new faults are inserted. Therefore, we prefer NHPP models. As we do not a priori know the exact nature of fault data (except that it can be described as NHPP), we have to select several NHPP models to ensure covering each type of them (Table V).

➢ *Step 3: Select parameter estimation technique*

For estimating parameters of SRGMs, we will use Maximum Likelihood Estimation (MLE) technique [18] because MLE fulfills most of the properties we favor, such as asymptotic normality, robustness and consistency. Besides, MLE simultaneously estimates model parameters and provides to easily derive confidence intervals.

➢ *Step 4: 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 correspondences 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. Besides, commonly used Mean Square Error (MSE) is selected [19].

➢ *Step 5: Select best model*

The SRGM with the smallest AIC, BIC and MSE is selected as best fitting model.

TABLE V. OVERVIEW OF THE SELECTED NHPP MODELS

Model	Mean value function	Class/Family	
Goel-Okumoto (G-O)	$\mu(t) = a(1-e^{-bt}) \quad b>0$	exponential class	finite failure model
Generalized Goel-Okumoto (GGO)	$\mu(t) = a(1-e^{-bt^c})$ $a>0, b>0, c>0$	weibull class	finite failure model
Delayed S-Shaped (D-S)	$\mu(t) = a\left(1-(1+bt)e^{-bt}\right)$ $a>0, b>0$	gamma class	finite failure model
Log-Power (LP)	$\mu(t) = a\ln^b(1+t) \quad t \geq 0$	power family	finite failure model
Duane (D)	$\mu(t) := a \cdot t^b$	power family	infinite fail.
Musa-Okumoto (M-O)	$\mu(t) = \frac{1}{\theta}\ln(\lambda_0 \theta t + 1)$	geometric family	infinite fail.

V. CASE STUDY

This section demonstrates and validates the LC testing approach and analyzes its characteristic features, including an experimental comparison with FR testing.

A. System Under Consideration, Setup and Results

SUC is a large commercial web portal with 53K LOC (lines of code): ISELTA [20] ("Isik's System for Enterprise-Level Web-Centric Tourist Applications"). ISELTA enables travel and tourist enterprises, e.g., hotel owners, to create their own individual search & service offering masks. These masks can be embedded in an existing homepage as an interface between customers and system. Customers can then use those masks to select and book services, e.g., hotel rooms, rental cars, etc. See Fig. 6 for a screenshot of ISELTA.

For testing a sub-system of ISELTA, a hierarchical set of ESGs has been constructed (Fig. 7). The main (or top level) ESG contains three vertices that are refined by ESGs of the 2nd layer. One of these ESGs, "provider account", is also a compound vertex and contains two sub-vertices, "edit profile" and "edit hotel", as its refinements. The ESG for "login as provider – quick" can be seen in Fig. 8.

Following figures are supposed to give an impression about the size and efforts of the case study:

- 7 different ESG with totally 73 vertices and 207 edges have been constructed.
- More than 60000 different tests have been generated and executed.
- 3 parties of tests for each approach have been generated subject to the length of ES covered (2, 3, and 4).

Figure 6. Screenshot of ISELTA

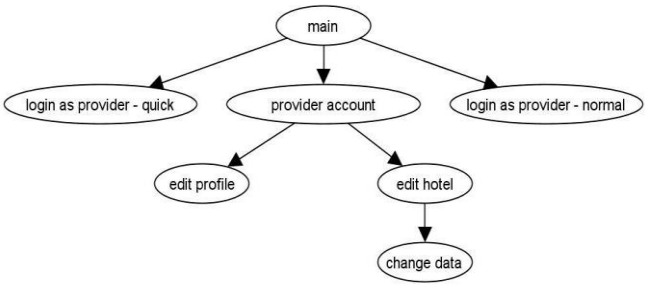

Figure 7. Hierarchical structure of the ESG used in the case study

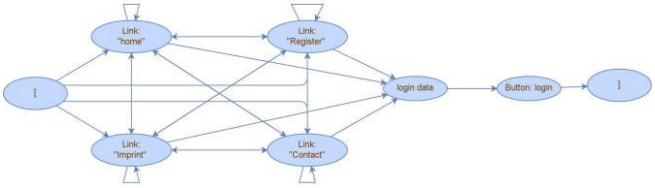

Figure 8. Refinement of vertex "Login as Provider - quick".

TABLE VI. RESULTS OF THE CASE STUDY

	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	77*	7005	7082	31+0
∑	120	50904	51024	39	83	9325	9408	31

* calculated on the basis of LC_{simple}

TABLE VII. NUMBER OF EVENTS TO BE EXECUTED

	FR			LC			
length	CES	FCES	∑	CES	FCES	∑	saving
2	353	17504	17857	341	3766	4107	77 %
3	1589	73053	74642	1430	13353	14783	80 %
4	9580	343969	353549	8983*	63171	72154	80 %
∑	11522	434526	446048	10754	80290	91044	80 %

* calculated on the basis of LC_{simple}

To support the case study, LC strategy (explained in Section III) has been implemented and integrated in a tool called *Test Suite Designer* (*TSD*). Within TSD, hierarchical ESGs have been modeled. Vertex annotations refer to source code executing the underlying event within a separate test execution environment. This enables to automatically generate test scripts along the calculated test sequences. Visit *http://quebec.upb.de/compsac2011.pdf* [17] for a detailed description of the algorithms and models used in the case study. Based on TSD, CESs and FCESs have been generated covering event sequences of length 2, 3 and 4. The results of the case study are summarized in Table VI.

CESs following the LC approach have been constructed on the basis of LC_{opt}. As covering test sequences of length 4 took unacceptable long time due to the complexity of the test generation algorithm, CESs covering length 4 sequences have been generated on the basis of LC_{simple}. Table VI includes also the results of the analysis for the case that the hierarchical ESG is resolved completely (thus, following the FR approach) before CESs and FCESs are generated. It can easily be seen that the number of CESs of our 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 are shown in Table VII. In total, the new approach, LC strategy, reduced the number of events by 80%. Surprising is the fact that also 80 % of the faults could be detected. In other words, for about 20% of the previous test effort about 80% of the faults could be detected!

Fig. 9 shows the number of test cases for event sequences of increasing length. Fig. 10 shows it on a logarithmic scale. The fact that the curve forms an almost straight line reflects the circumstance that the number of test cases grows exponentially with increasing length. The same holds for the number of events (not shown due to the lack of space). This is an important observation as this enables to perform a raw estimation of the number of test cases for increasing sequence length without prior construction of the sequences. Note that construction of sequences of higher length is very time consuming. This raw estimation can be used for reliability estimations to decide whether to increase the sequence length to detect additional faults.

B. *Analysis of the Results*

As described in Section IV, the results of the experiment will be carried out in five steps.

Step 1: As we have no time units given, the cumulative number of test cases (instead of points in time) and number of faults detected by event sequences of length 2, 3 and 4 (instead of number of failures detected in a time interval) are used for the reliability determination of software. According to Table VI, the grouped data consisted of cumulative number of test cases and has three groups which are defined by: [0, 587[, [587, 2326[,2326, 9408[, i.e., "times" t_1=587, t_2=2326, t_3=9408 with y_1=29, y_2=2, y_3=0 as failures per interval. Note that t_2 is calculated by $t_2 = t_1$ + *number of test cases covering length 3*, that is, t_2=587+1739 and so on.

Step 2: To decide whether or not Poisson type models can be used in this study, a K-S test is applied with following hypothesis:

H_0 : Cumulative number of failures comes from Poisson distribution

H_1 : Cumulative number of failures does not come from Poisson distribution

The results of the K-S test are given in Table VIII. Table VIII indicates that the cumulative number of failures follows Poisson distribution with mean parameter = 30.333 since p-value (0.709) is greater than 0.05.

Step 3: Fig. 11 shows predictions of mean-value and 95% confidence intervals for each SRGM at time t_3. According to Fig. 11, *G-O* and *D-S* model gave an exact prediction for the cumulative number of failures at time t_3. *M-O* model has given worst performance. This model has estimated that there were approximately 4 additional failures until time t_3., i.e., 4 additional failures by executing 7082 test cases covering length 4. One reason might be that failure intensity does not decrease exponentially with the expected number of failure experienced for this data.

Step 4: Fig. 12 shows the Goodness of Fit measures.

Step 5: It can easily be seen that *G-O* gives the best estimation in all Goodness of Fit measures since it has the smallest AIC, BIC, MSE values (the smaller the better).

To compare the reliability of the LC approach with the FR approach, the reliability of the FR approach has also been calculated which is given as R_{FR}=0.99870. Table IX shows the reliability measures of LC and FR approach for each reliability model. Table IX clearly indicates that there is no significant difference between the reliability of FR (R_{FR}=0.99876) and LC (R_{LC} = 0.99940) approach.

C. *Result in a nutshell*

In the case study, 31 faults were detected using the LC approach and 39 faults using the FR approach. Thus, LC detected 20% less faults. However, the new approach reduced the test effort by about 80%. The follow-on reliability analysis found out that the LC approach leads to an even slightly better reliability level as the FR approach.

TABLE VIII. ONE-SAMPLE KOLMOGOROV SMIRNOV TEST

	Cumulative Number Of Failures
Poisson Parameter Mean	30.333
Kolmogorov-Smirnov Z	0.701
p-value (2-tailed)	0.709

TABLE IX. RELIABILITY MEASURES

Models	G-O	D-S	D	GGO	LP	M-O
R_{LC}	**0.99940**	0.99998	0.88786	0.60851	0.17454	0.00465
R_{FR}	**0.99876**	0.99997	0.99997	0.63186	0.93659	0.03375

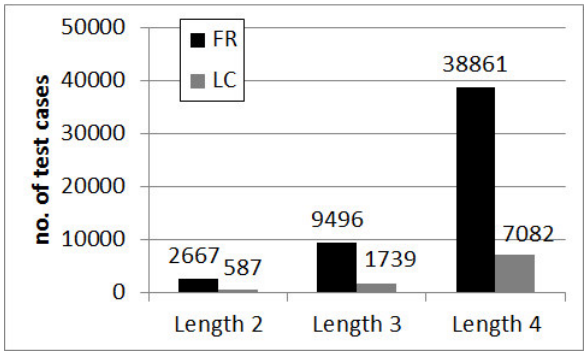

Figure 9. Number of test cases

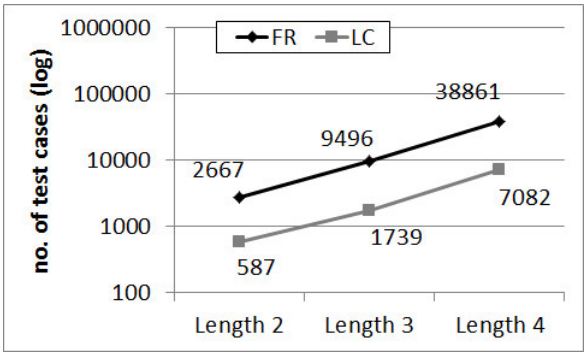

Figure 10. Number of test cases on a logarithmic scale

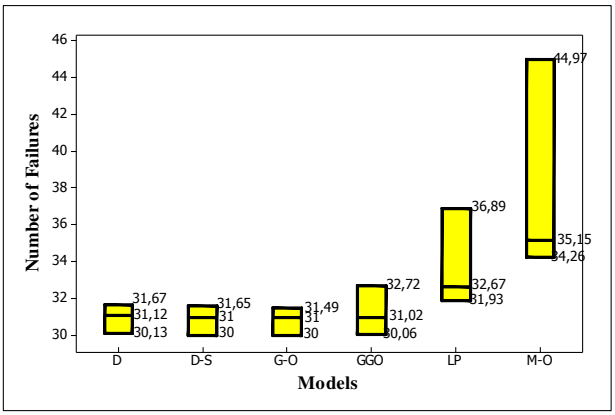

Figure 11. Predictions of Mean-Value and 95 % Confidence Intervals at time t_3 (thus, for length 4 testing)

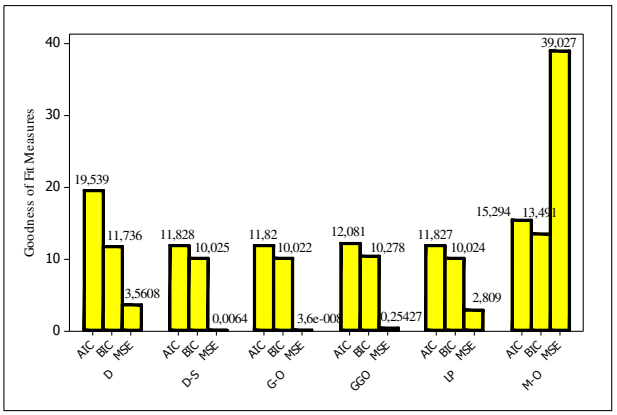

Figure 12. Goodness of Fit Measures

D. Threats to Validity

In general, while testing a system by a model-based technique, the tester assumes that the underlying model is correct and complete with respect to the considered entities. The same holds for our case study. By analyzing ESG models we could detect merely faults on events and their order. Other types of faults, for example in database interactions, are not at the focus even if they might be detected by chance. Data dependencies are also not considered. This can be seen as oversimplifying and thus restrictive; however, the number and severity of the faults detected indicates the strength of the approach (see [4], [5] for further examples of the fault detection effectiveness of ESG approach).

Another concern is about the transferability of the here achieved results to other systems or models as these results heavily depend on the given SUC, its development process and on many more factors. Furthermore, in our case only one system, namely ISELTA, is tested based on one large component. However, we believe that testing other components would lead to a similar effect of the test suite reduction.

In addition, reliability estimations heavily depend on the given fault data. Thus, the reliability growth models that are applicable to a given data set might turn out to be not applicable to another data set. The reliability estimations as performed in Section IV clearly demonstrate, however, the applicability of those models. Therefore, the reliability analysis should not be left out; it provides also a reasonable approach to define the point in time when to stop testing.

VI. CONCLUSIONS

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. Some of the model-based testing techniques allow creating a hierarchy of models since the state space of complex systems can be very large to be modeled. Nevertheless, the hierarchy has to be resolved before test cases are generated. This can, however, lead to an unfeasible large number of test cases. Thus, the deeper the model hierarchy, the more time and labor consuming are the test case generation and test execution.

This paper introduced the *layer-centric* (*LC*) testing approach based on event sequence graphs (ESG) to overcome these problems. LC makes use of the hierarchical structure of the model by generating test cases based on the single models so that the test generation and test effort will reasonably be reduced. A case study demonstrated and validated the approach, and analyzed its characteristics features.

The result is surprising: In our case study, compared to the full-resolution (FR) testing, the LC testing reduced the test suites by 80 % at a fault detection rate of 80 %. This fact encourages performing LC testing, especially in cases where only a small number of test cases can be afforded, e.g., in smoke testing.

Finally, the fault detection ability of the test suites generated by LC testing has been compared with the fault detection ability of the test suites generated by the FR testing by means of a reliability theoretical analysis. This analysis justifies the conclusion that LC testing leads to a reliability level that is comparable to, even slightly better than, the one achieved by FR testing. The reliability analysis points out also the fact that it is advisable to estimate the expected number of failures prior to increasing the test sequence length since the expected reliability growth could not justify the additional test costs.

Therefore, the reduction of test costs for increasing the test cases of length >2 by more efficient algorithms will form our future work.

REFERENCES

[1] IEEE, "IEEE Standard Glossary of Software Engineering Terminology," *IEEE Std 610.12-1990*, 1990.

[2] Object Management Group, Unified Modeling Language (UML), *http://www.omg.org*

[3] F. Belli and C.J. Budnik, "Test Minimization for Human-Computer Interaction", International Journal of Applied Intelligence, vol. 26(2), Springer, pp. 161-174, 2007.

[4] F. Belli , N. Güler, and M. Linschulte, "Are longer test sequences always better?- AReliability Theoretical Analysis", 4th IEEE Intern. Conf. on Secure Software Integration and Reliability Improvement, SSIRI 2010, pp. 78-85 , 2010.

[5] F. Belli, C.J. Budnik, and L. White, "Event-based modeling, analysis and testing of user interactions: Approach and case study," Journal of Software Testing, Verification and Reliability (STVR), vol. 16(1), John Wiley & Sons, Ltd. pp. 3-32, 2006.

[6] A. M. Memon, M.E. Pollack, and M. L. Soffa, "Hierarchical GUI Test Case Generation Using Automated Planning," IEEE Transactions on Software Engineering 27(2): 144-155, 2001.

[7] A. C. R. Paiva, N. Tillmann, J. C. P. Faria, and R.F.A.M. Vidal, "Modeling and Testing Hierarchical GUIs", in Proc. of 12th Intern. Workshop on Abstract State Machines, Paris France, pp. 8-11, 2005.

[8] A. A. Andrews, J. Offutt, and R.T. Alexander, "Testing Web applications by modeling with FSMs", Software and Systems Modeling 2005, vol. 4(3), pp. 326-345, 2005.

[9] H. Reza, S. Endapally, and E. Grant, "A Model-Based Approach for Testing GUI Using Hierarchical Predicate Transition Nets," Intern. Conf. on Information Technology (ITNG'07), pp. 366-370, 2007.

[10] "IEEE Recommended Practice on Software Reliability," IEEE STD 1633-2008, pp.c1-72, 2008.

[11] "Guidance on Software Aspects of Dependability (IEC 56/1349A/CD:2009)," DIN REC 62628, March, 2010.

[12] M. R. Lyu, "Handbook of Software Reliability Engineering". McGraw-Hill, 1996.

[13] A. Arcuri, "Longer is Better: On the Role of Test Sequence Length in Software Testing", ICST 2010, Third International Conference on Software Testing, Verification and Validation, 2010.

[14] D.B. West, "Introduction to Graph Theory", Prentice Hall, 1996.

[15] R. Burkard, M. Dell'Amico, S. Martello, "Assignment Problems", Society for Industrial and Applied Mathematics, Philadelphia, 2009.

[16] H. Thimbleby, "The directed Chinese Postman Problem", Journal of Software – Practice and Experience, vol. 33(11), pp. 1081-1096, 2003.

[17] *http://quebec.upb.de/compsac2011.pdf* for a complete documentation of algorithms and models

[18] A. Birolini, Reliability Engineering: Theory and Practice, Springer, 2007.

[19] K. Shibata, K. Rinsaka and T. Dohi, "Metrics-Based Software Reliability Models Using Non-homogeneous Poisson Processes", Proc. of the 17th Intern. Symp. on Software Reliability Engineering, IEEE Computer Society, pp. 52-61, 2006.

[20] ISELTA, *http://www.iselta.de/*