Test Cost Reduction for Interactive Systems

Fevzi Belli, Christof J. Budnik

University of Paderborn, Warburger Str. 100, 33098 Paderborn, Germany
{belli, budnik}@adt.upb.de

Abstract. A model-based approach for minimization of test sets for human-computer interactions is introduced. Test cases are efficiently generated and selected to cover the behavioral model of the system under test (SUT) and its fault model that is constructed by complementing the original model. Results known from state-based conformance testing and graph theory are used and extended to construct algorithms for minimizing the test sets, considering structural features of the SUT.

1 Introduction

Testing is the traditional validation method in the software industry. There is no justification, however, for any assessment on the correctness of the SUT based on the success (or failure) of a single test, because there can potentially be an infinite number of test cases, even for very simple programs. To overcome this shortcoming of testing, formal methods have been proposed, which introduce models that represent the relevant features of the SUT. The modeled, relevant features are either functional behavior or the structural issues of the SUT, leading to *specification-oriented testing* or *implementation-oriented testing*, respectively. This paper is on specification-oriented testing; i.e., the underlying model represents the system behavior interacting with the user's actions. The system's behavior and user's actions will be viewed here as *events,* more precisely, as *desirable events* if they are in accordance with the user expectations. Moreover, the approach includes modeling of the faults as *undesirable events* as, mathematically spoken, a complementary view of the behavioral model.

Based on [Be01], this paper introduces a novel, graphical representation of both the behavioral model and the fault model of the SUT. Algorithms are introduced for the coverage of these models by a minimal set of test cases (*minimal spanning set for coverage testing*). The next section summarizes the related work before Section 3 introduces the fault model and the test process. The optimization of the test suite is discussed in Section 4. Section 5 considers the structure of the SUT to avoid unnecessary and/or infeasible tests. The achieved results are summarized in Section 6. Section 7 concludes the paper and sketches the research work planned.

2 Related Work

Methods based on finite-state automata have been used for almost four decades for the specification and testing of system behavior, e.g., for specification of software systems [Ch78], as well as for conformance and software testing [Bi00, ADL91, Sa89, OSA03]. Also, the modeling and testing of interactive systems with a state-based model has a long tradition [Pa69, JNC03, SS97, WA00]. These approaches analyze the SUT and model the user requirements to achieve sequences of *user interaction* (*UI*), which then are deployed as test cases. [WA00] introduced a simplified state-based, graphical model to represent UIs; this model has been extended in [Be01] to consider not only the desirable situations, but also the undesirable ones. This strategy is quite different from the combinatorial ones, e.g., *pairwise testing,* which requires that for each pair of input parameters of a system, every combination of these parameters' valid values must be covered by at least one test case. It is, in most practical cases, not feasible [TL02] to test UIs.

A similar fault model as in [Be01] is used in the mutation analysis and testing approach which systematically and stepwise modifies the SUT using mutation operations [DLS78]. Although originally applied to implementation-oriented unit testing, mutation operations have also been extended to be deployed at more abstract, higher levels, e.g., integration testing, state-based testing, etc. [DMM01]. Such operations have also been independently proposed by other authors, e.g., "state control faults" for fault modeling in [BP94], or for "transition-pair coverage criterion" and "complete sequence criterion" in [OSA03]. However, the latter two notions have been precisely introduced in [Be01] and [WA00], respectively, earlier than in [OSA03]. A different approach, especially for graphical user interface (GUI) testing, has been introduced in [MPS00]; it deploys methods of knowledge engineering to generate test cases, test oracles, etc., and to deal with the test termination problem. All of these approaches use some heuristic methods to cope with the state explosion problem.This paper also presents a method for test case generation and test case selection. Moreover, it addresses test coverage aspects for test termination, based on [Be01], which introduced the notion of "minimal spanning set of complete test sequences", similar to "spanning set", that was also later discussed in [MB03]. The present paper considers existing approaches to optimize the round trips, i.e., the Chinese Postman Problem [ADL91], and attempts to determine algorithms of less complexity for the spanning of walks, rather than tours, related to [We96, NT81].

3 Fault Model and Test Process

The SUT we use in the examples in this paper is a control terminal of a marginal strip mower (Figure 1) which controls a marginal strip mower (RSM 13), that takes optimum advantage of mowing around guide poles, road signs and trees, etc. Operation is effected either by the power hydraulic of a light truck, or by the front power take-off. Further buttons on the control desk (Figure 1) simplify the operation, so that, e.g., the mow head returns to working position or to transport position when a button is pressed.

Figure 1: The example vehicle (RSM 13) and its control desk

This work uses *Event Sequence Graphs (ESG)* for representing the system behavior and, moreover, the facilities from the user's point of view to interact with the system. Basically, an *event* is an externally observable phenomenon, such as an environmental or a user stimulus, or a system response, punctuating different stages of the system activity.

3.1 Preliminaries

Definition 1. An Event Sequence Graph ESG=(V,E) is a directed graph with a finite set of *nodes* (*vertices*) $V \neq \emptyset$ and a finite set of *arcs* (*edges*) $E \subseteq V \times V$.

For representing user-system interactions, the nodes of the ESG are interpreted as events. The operations on identifiable components of the UI are controlled/perceived by input/output devices, i.e., elements of windows, buttons, lists, checkboxes, etc. Thus, an event can be a user input or a system response; both of them are elements of *V* and lead interactively to a succession of user inputs and system outputs.

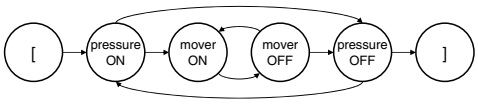

Figure 2. An excerpt of the ESG of the RSM 13 illustrating the interaction between the cutting unit and the pressure on bottom with "["as entry and "]" as exit

Definition 2. Let *V*, *E* be defined as in Def. 1. Then any sequence of nodes $\langle v_0,\ldots,v_k\rangle$ is called an (legal) *event sequence* (*ES*) if $(v_i, v_{i+1}) \in E$, for $i=0,\ldots,k-1$.

Furthermore, α (*initial*) and ω (*end*) are functions to determine the initial node and end node of an ES, i.e., $\alpha(ES)=v_0$, $\omega(ES)=v_k$. Finally, the function *l*(*length*) of an ES determines the number of its nodes. In particular, if $l(ES)=1$ then $ES=\langle v_i\rangle$ is an ES of length 1. An $ES=\langle v_i, v_k\rangle$ of length 2 is called an *event pair* (*EP*). The assumption is made that there is an ES from the single node ε to all other nodes, and from all nodes there is an ES to the single node γ ($\varepsilon, \gamma \notin V$). ε is called the *entry* and γ is called the *exit* of the ESG.

The entry and exit, represented by '[' and ']', respectively, are not included in V. They enable a simpler representation of the algorithms to construct minimal spanning test case sets (Section 4).

Definition 3. An ES is called a *complete ES* (*Complete Event Sequence*, *CES*), if $\alpha(ES)=\varepsilon$ is the entry and $\omega(ES)=\gamma$ is the exit.

CESs represent *walks* from the entry "*[*" of the ESG to its exit "*]*" and are interpreted as desirable events that fulfill user expectations.

3.2 Fault Model and Test Terminology

Definition 4. For an *ESG=(V, E)*, its *completion* is defined as $\hat{E}SG=(V,\hat{E})$ with $\hat{E}=V\times V$.

Definition 5. The *inverse* (or *complementary*) *ESG* is then defined as $\overline{ESG}=(V,\overline{E})$ with $\overline{E}=\hat{E}\setminus E$ (\: set difference operation).

Note: Entry and exit are not considered while constructing the $\overline{ESG}$.

Definition 6. Any EP of the *ESG* is a *faulty event pair* (*FEP*) for ESG.

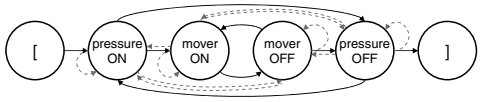

Figure 3. The completion $\hat{E}SG$ of Figure 2

Definition 7. Let $ES=\langle v_0,\ldots,v_k\rangle$ be an event sequence of length $k+1$ of an ESG and $FEP=\langle v_k,v_m\rangle$ a faulty event pair of the according *ESG*. The concatenation of the ES and FEP forms then a *faulty event sequence* $FES=\langle v_0,\ldots,v_k,v_m\rangle$.

Definition 8. An *FES* will be called *complete* (*Faulty Complete Event Sequence*, *FCES*) if $\alpha(FES)=\varepsilon$ is the entry. The ES as part of a FCES is called a *starter*.

3.3 Discussion of the Model

ESGs can be viewed as a simplification of the finite state automata; they have the same computational power. Complementing the ESG is quite simple and theoretically secure because of the neat features of the type-3 languages. The tradeoff for the simplification through ESG modeling is that it neglects the states of the SUT and the hierarchical levels of the user interactions. Generation of test cases, which needs information about the inner behavior of the system, might become difficult, e.g., to check that a save operation will not be executed if the loaded file is write-protected. Presentation of such situations

is generally possible, but might become tedious because of the likely numerous combinations of different values of corresponding flags, which could have been set or reset in different menus. Remember that it is not realistic to suppose that a single test method will serve as a "silver bullet", coping with all kinds of faults.

3.4 Test Process

Definition 9. A *test case* is an ordered pair of an input and expected output of the SUT. Any number of test cases can be compounded to a *test set* (or, a *test suite*).

Once a test set has been constructed, tests can be run applying the test cases to the SUT. If it behaves as expected, the SUT *succeeds* the test, otherwise it *fails* the test. The approach introduced in this paper uses event sequences, more precisely CES, and FCES, as test inputs. If the input is a CES, the SUT is supposed to proceed it and thus, to succeed the test. Accordingly, if a FCES is used as a test input, a failure is expected to occur. The latter case represents an exception that must be properly handled by the system, i.e., the SUT is supposed to refuse the proceeding and produce a warning. The test process is sketched in Algorithm 1.

Algorithm 1. Test Process

n:= number of the functional units (modules) of the system that fulfill a well-defined task
length:= required length of the test sequences
`FOR` function 1 `TO` n `DO`
Generate appropriate ESG and $\overline{\text{ESG}}$
`FOR k:=2 TO length DO` //Section 4.2
Cover all ESs of length k by means of CESs subject to minimizing the number and total length of the CESs //Section 4.1
Cover all FEPs of by means of FCESs subject to minimizing the total length of the FCESs //Section 4.3
Apply the test set to the SUT.
Observe the system output to determine whether the system response is in compliance with the expectation.

To determine the point in time in which to stop testing, a criterion is necessary to systematize the test process and to judge the efficiency of the test cases. The approach converts this problem into the *coverage of the ES and FES of length k of the* $\overline{ESG}$. The test costs are given by the minimized total length of the CESs and FCESs. The length of the ESs can be increased stepwise. This enables a scalability of the test costs which are proportional to the length of the ESs.

4 Minimizing the Spanning Set

The union of the sets of CESs of minimal total length to cover the ESs of a required length is called *Minimal Spanning Set of Complete Event Sequences (MS^{2}CES).*

If a CES contains all EPs at least once, it is called an *entire walk*. A legal entire walk is *minimal* if its length cannot be reduced. A minimal legal walk is *ideal* if it contains all EPs exactly once. Legal walks can easily be generated for a given ESG as CESs, respectively. Entire walks are very convenient if the *reset* requirement is not fulfilled. It is not, however, always feasible to construct an entire or ideal walk.

4.1 An Algorithm to Determine Minimal Spanning Complete Event Sequence

The determination of MS^2CES represents a derivation of the *Directed Chinese Postman Problem (DCPP)*, which has been studied thoroughly, e.g., in [ADL91, Th03]. The MS^2CES problem introduced here is expected to have a lower complexity grade, as the edges of the ESG are not weighted, i.e., the adjacent vertices are equidistant. In the following, some results are summarized that are relevant to calculate the test costs and enable scalability of the test process.

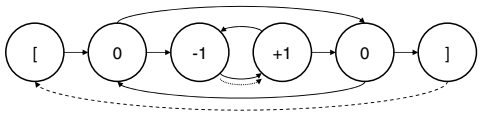

Figure 4: Transferring walks into tours and balancing the nodes

For the determination of the set of minimal tours that covers the edges of a given graph, the algorithm described in [Th03] requires this graph be strongly connected. This can be reached for any ESG through an additional edge from the exit to the entry, which is interpreted as resetting the system. The idea of transforming the ESG into a strongly connected graph is depicted in Figure 4 as a dashed arc. The figures within the vertices indicate the balance of these vertices as the difference of the number of outgoing edges and the number of the incoming edges. These balance values determine the minimal number of additional edges from "+" to "-" that will be identified by searching the all-shortest-path and solving the optimization problem [AMO93] by the Hungarian method [Kn93]. The required additional edge for the ESG in Figure 4 is represented as a dotted arc. The problem can then be transferred to the construction of the Euler tour for this graph [We96]. Each occurrence of the ES=*]]* in the Euler tour identifies another separate test case. The algorithm to determine minimal spanning set of complete event sequences (MS^2CES) consists of three sections:

- *Determination of all-shortest-paths* by Floyds algorithm with the complexity $O(|V|^3)$ [We96]. However, because the ESG is a non-weighted digraph, the complexity can be decreased by using the Breadth-First-Search (BFS) down to $O(|V|\cdot|E|)$. This results from the $|V|$ loop invocations of the BFS algorithm which determines the shortest path from one node to all the others in $O(|E|)$ because the ESG is connected and $|E|>|V|+1$. In worst case when the ESG is fully connected then $|E|$ equals $|V|^2$ so that the complexity is the same as Floyds algorithm.
- The *optimizing problem*, also known as bipartite weighted matching problem, which is solved in accordance with [Kn93] by the Hungarian method, with the complexity $O(|V|^3)$.

- *Computation of an Euler tour* with the complexity of O(|V|·|E|) [We96].

To sum up, the MS^{2}CES can be solved in O(|V|^3) time.

4.2 Generating Event Sequences with Length > 2

A phenomenon in testing interactive systems is that faults can be frequently detected and reproduced only in some context. This makes a test sequence of a length>2 necessary since repetitive occurrences of some subsequences are needed to cause an error to occur/re-occur. Therefore an ESG can be transformed into a graph in which the nodes can be used to generate test cases of length > 2, in the same way that the nodes of the original ESG are used to generate EPs and to determine the appropriate MS^{2}CES. The common valid of this approach is given by Algorithm 2.

Algorithm 2. Generating ESs and FESs with length > 2
Input: *ESG=(V, E)*; $\varepsilon = [$, $\gamma =]$, *ESG'=(V', E')* with $V' = \emptyset$, $\varepsilon' = [$, $\gamma' =]$;
Output: *ESG'=(V', E')*, $\varepsilon' = [$, $\gamma' =]$;

```
FOR EACH (i, j) ∈ E with (i != ε) AND (j != γ) DO
    add_node(ESG', (ES(ESG, i) ⊕ ω (ES(ESG, j)));
    remove_arc(ESG, (i , j));
FOR EACH i ∈ V' with (i != ε') AND (i != γ') DO
    FOR EACH j ∈ V' with (j != ε') AND (j != γ') DO
        IF(ES(ESG', i) ⊕ ω (ES(ESG', j)) = α (ES(ESG', i)) ⊕ (ES(ESG', j)) THEN
            add_arc(ESG', (i, j))
FOR EACH (k, l) ∈E with k = ε DO
    IF(ES(ESG', i) = ES(ESG, l) ⊕ ω (ES(ESG', i)) THEN
        add_arc(ESG', (ε', i));
FOR EACH (k, l) ∈ E with l = γ DO
    IF (ES(ESG', i) = α (ES(ESG', i)) ⊕ ES(ESG, k) THEN
        add_arc(ESG', (i, γ'));
RETURN ESG';
```

Therein the notation ES(ESG, i) represents the identifier, e. g., *AB*, of the node *i* of the *ESG*. This identifier can be concatenated with another identifier ES(ESG, j) of the node *j*, e.g., *CD*. This is represented by *AB* ⊕ *CD*, or ES(ESG, i) ⊕ ES(ESG, j), resulting in the new identifier *ABCD*. Note that the identifiers of the newly generated nodes to extend the ESG will be made up using the identifiers of the existing nodes. The function add_node() inserts a new ES of length *k*. Following this step, a node *u* is connected with a node *v* if the last *n-1* events that are used in the identifier of *u* are the same as the first *n-1* events that are included in the identifier of *v*. The function add_arc() inserts an arc, connecting *u* with *v* in the ESG. The pseudo nodes *[*, *]* are connected with all the extensions of the nodes with which they were connected before the extension.

In order to avoid traversing the entire matrix, arcs which are already considered are to be removed by the function `remove()`. Apparently, the Algorithm 2 has a complexity of $O(|V|^2)$ because of the nested FOR-loops to determine the arcs in the *ESG'*. The algorithm to determine MS^2CES can be applied to the outcome of the Algorithm 2, i.e., to the extended ESG, to determine the MS^2CES for $l(ES) > 2$.

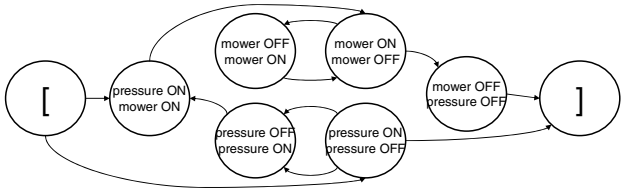

Figure 5: Extended ESG' for covering ESs of length=3

Figure 5 depicts the new generated ESG' of the ESG in Figure 2 for covering ESs of length=3. In case ESs of length=4 are to be generated, the extended graph must be extended another time using the same algorithm.

4.3 Minimal Spanning Set for the Coverage of Faulty Event Sequences

The union of the sets of FCESs of the minimal total length to cover the FESs of a required length is called *Minimal Spanning Set of Faulty Complete Event Sequences (MS^2FCES)*. In comparison to the interpretation of the CESs as legal walks, *illegal walks* are realized by FCESs that never reach the exit. An illegal walk is *minimal* if its starter cannot be shortened. Assuming that an ESG has n nodes and d arcs as EPs to generate the CESs, then at most $u:=n^2-d$ FCESs of minimal length, i.e., of length 2, are available. Those FCESs emerge when the node(s) after entry is (are) followed immediately by a faulty input. The number of FCESs is precisely determined by the number of FEPs, which are of constant length 2; thus, they cannot be shortened. It remains to be noticed that only the starters of the remaining FEPs can be minimized, e.g., using the algorithm given in [ED59].A further algorithm to generate FESs of length > 2 is not necessary because such faulty sequences will be constructed through the concatenation of the appropriate starters with the FEPs. Instead of that the next chapter shows how the structure of interactive systems can be algorithmically exploited to save infeasible and/or unnecessary test cases.

5 Exploiting the Structural Features

The approach has been applied to the testing and analysis of the GUIs of different kind of systems, leading to a considerable amount of practical experience [Be01]. A great deal of test effort could be saved considering the structural features of the SUT. Thus, there is further potential for the reduction of the cost of the test process.

Analysis of the structure of the GUIs delivers the following features:

- Windows of commercial systems are nowadays mostly hierarchically structured, i.e., the root window invokes children windows that can invoke further (grand) children, etc.
- Some children windows can exist simultaneously with their siblings and parents; they will be called *modeless* (or *non-modal*) windows. Other children, however, must "die", i.e., close, in order to resume their parents (*modal* windows).

Figure 6 represents these window types as a "family tree". In this tree, a unidirectional edge indicates a modal parent-child relationship. A bidirectional edge indicates a modeless one.

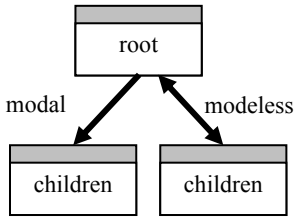

Figure 6: Modal windows vs. modeless windows

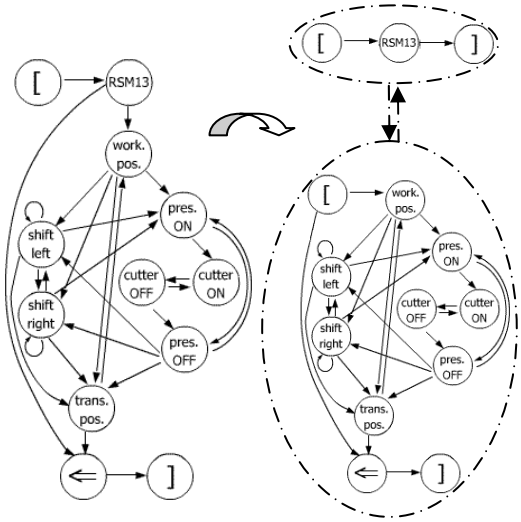

Figure 7: The ESG of the control terminal of Figure 1 and its modified ESG taken the modality feature into account.

Because modal windows must be closed before any other window can be invoked, it is not necessary to consider the FESs of the parent and children. This is true only for the FCESs and MS^2FCES as test inputs considering the structure information might impact the structure of the ESG, but not the number of the CESs and MS^2CESs as test inputs.

Thus, similar to the strong-connectedness and symmetrical features [SS97], the modality feature is extremely important for testing since it avoids unnecessary test efforts. Figure 7 represents the modified ESG. The modification, which separates the event `RSM13` takes the modality into account that avoids unnecessary FEPs.

6. Tool Support and Validation

The determination of the MS^2CESs/MS^2FCESs can be very time consuming when carried out manually. For that purpose the tool "GenPath" is developed by our group to input and process the adjacency matrix of the ESG. The user can, however, input several ESGs which are refinements of the vertices of a large ESG to be tested.

For a comprehensive testing, several strategies have been developed with varying characteristics of the test inputs, i.e., stepwise and scalable increasing and/or changing the length and number of the test sequences, and the type of the test sequences, i.e., CES- and FCESs-based, and their combinations. Following could be observed: The test cases of the length 4 were more effective in revealing dynamic, intricate faults than the test cases of the lengths 2 and 3. Even though more expensive to be constructed and exercised, they are more efficient in terms of costs per detected fault. Further on the CES-based test cases as well as the FCES-based cases were effective in detecting faults.

Due to the lack of space, the experiences with the approach are here very briefly summarized. This can be, however, found in [Be01] and [BNB04]. To sum up the test process, one student tester carried out 238 tests semi-automatically and detected a total of 39 faults, including some severe ones (Table 1).

Table 1: Two of the detected faults of the RSM control terminal

No.	Faults Detected by the FCES
1.	The cutting unit can be activated without having any pressure on the bottom, which is very dangerous if pedestrians approach the working area (According to the dashed (faulty) arc from "pressure OFF" to "cutter ON" in Fig. 3).
2.	Keeping the button for shifting the mow head pushed and changing to another screen causes control problems of shifting: The mower head with the cutting unit cannot immediately be stopped in an emergency case.

In a second stage, the results of the research work for minimizing the spanning set of the test cases (MS^2CES and MS^2FCES) have been applied to the testing of the margin strip mower. Table 2 demonstrates that the minimization algorithm (Section 4) could save in average about 65 % of the total test costs, while the exploitation of the structural information (Section 5) of the SUT could further save up to almost 30 %.

Table 2 . Reducing the number of test cases

Length	#CES	# MS^2CES	Cost Reduction ES
2	40	15	62.5 %
3	183	62	66.1 %
4	549	181	67.0 %
Sum	772	258	65.2 %

Length	# MS^{2}FCES without structural information	# MS^{2}FCES with structural information	Cost Reduction MS^{2}FCES
2	75	58	22.7 %
3	167	218	35.7 %
4	487	292	40.0 %
Sum	729	568	32.8 %

7 Conclusions and Future Work

This paper has introduced an integrated approach to coverage testing of interactive systems, incorporating modeling of the system behavior with fault modeling and minimizing the test sets for the coverage of these models. The framework is based on the concept of "event sequence graphs (ESG)". Event sequences (ES) represent the human-computer interactions. Accordingly, the fault model can be exploited to detect faults in these interactions. An ES is complete (CES) if it produces desirable, well-defined and safe system functionality. The notion of faulty complete event sequences mathematically complements this view. The objective of testing is the construction of a set of CESs of minimal total length that covers all ESs of a required length. A similar optimization problem arises for the validation of the SUT under exceptional, undesirable situations which are modeled by faulty event sequences (FESs) and complete FESs (FCESs). The paper applied and modified algorithms known from graph theory to these problems. Furthermore, it was shown how the structure of interactive systems can be algorithmically exploited by a commercial test tool to reduce the test sets by infeasible and/or unnecessary test cases. In the case of *safety*, the threat originates from within the system due to potential failures and its spillover effects causing potentially extensive damage to its environment.

While some of the results of the analysis of the detected faults were in compliance with the expectations, other results were surprising, e.g., detection of modeling errors such as the identification of missing nodes that were forgotten, or illegally omitted during ESG modeling, although the fault model was not constructed to detect such faults. As a recommendation for practice the determination and specification of the CESs and FCESs should ideally be carried out during the definition of the user requirements, long before the system is implemented; the availability of a prototype would be very helpful. However, CESs and FCESs can also be produced incrementally at any later time, even during the test stage, in order to discipline the test process. Most of the studied SUTs do not consider the handling of the faulty events. They have only a rudimentary, if any, exception handling mechanism that mostly leads to a crash. The number of the exceptions that should be handled systematically, but have not been considered at all by the GUIs of the commercial systems is presumed to be an average of about 80%.

The goal for future work is to design defense actions, which is an appropriately enforced sequence of events, to prevent faults that could potentially lead to such failures. Further future work concerns cost reduction through automatic, or semiautomatic modification of a given ESG in order to consider modality of interaction structures.

Literature

[ADL91] A. V. Aho, A. T. Dahbura, D. Lee, M. Ü. Uyar, "An Optimization Technique for Protocol Conformance Test Generation Based on UIO Sequences and Rural Chinese Postman Tours", *IEEE Trans. Commun.* 39, pp. 1604-1615, 1991

[AMO93] R. K. Ahuja, T. L. Magnanti, J. B. Orlin, "Network Flows-Theory, Algorithms and Applications", Prentice Hall, 1993.

[Be01] F. Belli, "Finite-State Testing and Analysis of Graphical User Interfaces", Proc. 12th ISSRE, pp. 34-43, 2001

[Bi00] R.V. Binder, "*Testing Object-Oriented Systems*", Addison-Wesley, 2000

[BNB04] F. Belli, N. Nissanke, Ch. J. Budnik, "A Holistic, Event-Based Approach to Modeling, Analysis and Testing of System Vulnerabilities"; TR 2004/7, Univ. Paderborn, 2004

[BP94] G. V. Bochmann, A. Petrenko, "Protocol Testing: Review of Methods and Relevance for Software Testing", *Softw. Eng. Notes, ACM SIGSOFT*, pp. 109-124, 1994

[Ch78] Tsun S. Chow, "Testing Software Designed Modeled by Finite-State Machines", *IEEE Trans. Softw. Eng.* 4, pp. 178-187, 1978

[DLS78] R.A. DeMillo, R.J. Lipton, F.G. Sayward, "Hints on Test Data Selection: Help for the Practicing Programmer", *Computer* 11/4, pp. 34-41, 1978

[DMM01] M.E. Delamaro, J.C. Maldonado, A. Mathur, "Interface Mutation: An Approach for Integration Testing", *IEEE Trans. on Softw. Eng.* 27/3, pp. 228-247, 2001

[ED59] Edsger. W. Dijkstra, "A note on two problems in connexion with graphs.", Journal of Numerische Mathematik, Vol. 1, pp. 269-271, 1959

[JNC03] J. Jorge, N.J. Nunes, J.F. Cunha (Eds.), "Interactive Systems – Design, Specification, and Verification", LNCS 2844, Springer-Verlag, 2003

[Kn93] D.E. Knuth, "The Stanford GraphBase", Addison-Wesley, 1993

[MB03] M. Marré, A. Bertolino, "Using Spanning Sets for Coverage Testing", *IEEE Trans. on Softw. Eng.* 29/11, pp. 974-984, 2003

[MPS00] A. M. Memon, M. E. Pollack and M. L. Soffa, "Automated Test Oracles for GUIs", *SIGSOFT 2000*, pp. 30-39, 2000

[NT81] S. Naito, M. Tsunoyama, "Fault Detection for Sequential Machines by Transition Tours", *Proc. FTCS*, pp. 238-243, 1981

[OSA03] J. Offutt, L. Shaoying, A. Abdurazik, and Paul Ammann, "Generating Test Data From State-Based Specifications", *The Journal of Software Testing, Verification and Reliability, 13(1):25-53, Medgeh 2003*.

[Pa69] D.L. Parnas, "On the Use of Transition Diagrams in the Design of User Interface for an Interactive Computer System", Proc. 24th ACM Nat'l. Conf., pp. 379-385, 1969

[Sa89] B. Sarikaya, "Conformance Testing: Architectures and Test Sequences", *Computer Networks and ISDN Systems* 17, North-Holland, pp. 111-126, 1989

[SS97] R. K. Shehady and D. P. Siewiorek, "A Method to Automate User Interface Testing Using Finite State Machines", in *Proc. Int. Symp. Fault-Tolerant Computing FTCS-27*, pp. 80-88, 1997

[Th03] H. Thimbleby "The Directed Chinese Postman Problem", School of Computing Science, Middlesex University, London, 2003

[TL02] K. Tai, Y. Lei, "A Test Generation Strategy for Pairwise Testing", IEEE Trans. On Softw. Eng. 28/1, pp. 109-111, 2002

[We96] D.B. West, "Introduction to Graph Theory", Prentice Hall, 1996

[WA00] L. White and H. Almezen, "Generating Test Cases for GUI Responsibilities Using Complete Interaction Sequences", in Proc ISSRE, IEEE Comp. Press, pp. 110-119, 2000