Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2015) 23: 623 – 640
© TÜBİTAK
doi:10.3906/elk-1209-109

Model-based test case prioritization using cluster analysis: a soft-computing approach

Nida GÖKÇE[1,*], Fevzi BELLİ[2], Mübariz EMİNLİ[3], Bekir Taner DİNÇER[4]

[1]Department of Statistics, Faculty of Science, Muğla Sıtkı Koçman University, Muğla, Turkey
[2]Faculty of Computing Science, Electrical Engineering, and Mathematics, University of Paderborn, Paderborn, Germany
[3]Department of Computer Engineering, Faculty of Engineering, Haliç University, İstanbul, Turkey
[4]Department of Computer Engineering, Faculty of Engineering, Muğla Sıtkı Koçman University, Muğla, Turkey

Received: 24.09.2012 • **Accepted/Published Online:** 26.03.2013 • **Printed:** 30.04.2015

Abstract: Model-based testing is related to the particular relevant features of the software under test (SUT) and its environment. Real-life systems often require a large number of tests, which cannot exhaustively be run due to time and cost constraints. Thus, it is necessary to prioritize the test cases in accordance with their importance as the tester perceives it, usually given by several attributes of relevant events entailed. Based on event-oriented graph models, this paper proposes an approach to ranking test cases in accordance with their preference degrees. For forming preference groups, events are clustered using an unsupervised neural network and fuzzy c-means clustering algorithm. The suggested approach is model-based, so it does not necessitate the availability of the source code of the SUT. It differs from existing approaches also in that it needs no prior information about the tests carried out before. Thus, it can be used to reflect the tester's preferences not only for regression testing as is common in the literature but also for ranking test cases in any stage of software development. For the purpose of experimental evaluation, we compare the suggested prioritization approach with six well-known prioritization methods.

Key words: Test prioritization, model-based testing, event-oriented graphs, event sequence graphs, clustering algorithms, fuzzy c-means, neural networks

1. Introduction

As a means of quality assurance in the software industry, testing is one of the well-known analysis techniques [1–3]. However, since all possible test cases can potentially be infinite in number, there is no justification for any assessment of the correctness of the software under test (SUT) based on the success or failure of a single test case. To tackle this challenge, which concerns completeness of validation, some formal methods, which usually use models to visualize the desirable characteristics of the SUT, are proposed. Those characteristics are either related to functional behaviors or structural issues of the SUT: the former characteristics lead to specification-oriented testing and the latter characteristics lead to implementation-oriented testing. By using such a model, one can generate and select test cases as ordered pairs of test inputs and expected test outputs. To ascribe a measure to the effectiveness of a collection of test cases (i.e. the effectiveness of a test suite) in revealing faults [2,4], a coverage-oriented [5] adequacy criterion is used in this study since this criterion uses the ratio of the portion of the specification or code that is covered by the given test suite to the uncovered portion:

*Correspondence: nidagokce@yahoo.com

the higher the degree of test coverage is, the lower is the risk of having critical software artifacts that have not been sifted through.

The approach proposed in this paper is both specification-oriented and coverage-oriented. Software can be tested, either to verify that it is functioning as expected (positive testing) or to verify that it is not functioning in a way that is contrary to expectations (negative testing). The distinction between correct and faulty functioning of the SUT is referred to as the oracle problem in the literature [6,7]. In this regard, we represent the behavior of a system in interacting with the user's actions by means of event sequence graphs (ESGs) [6–8], in the sense of positive testing. In an ESG, desirable events are ones that are in accordance with the user expectations and, conversely, undesirable events are those that are discordant with the user expectations [6–9]. The obtained model is then analyzed to generate test cases for positive testing.

From the knowledge engineering point of view, testing is considered as a planning problem that can be solved using a goal-driven strategy [10], such that, given a set of operators, an initial state, and a goal state, the planner is expected to produce a sequence of operators by means of which the system can run from the initial state to the goal state. In relation to the testing problem described above, this means that an appropriate test sequence needs to be constructed upon the desirable, correct inputs. The test coverage problem then becomes finding the shortest path that visits each arc (e.g., node, arc-pair) at least once in a given ESG. Here, the above optimization problem is a generalization of the Chinese postman problem (CPP) [11]. Although there are numerous algorithms to solve the CPP, the algorithms given in [6,7] are different from the others in that they satisfy not only the constraint of a minimum total length of test sequences but also cover all the event pairs represented graphically. This brings about a substantial improvement in solving the test termination problem and thus constitutes one of the benefits of the proposed approach.

In particular, this article proposes a prioritized version of the mentioned test generation and optimization algorithms on the basis of the well-known "divide and conquer" principle. It is a fact that prioritization should be done in a way that schedules the test process, i.e. to meet the needs and the preferences of the test management that commonly aims to minimize the test budget. However, the SUT and software objects (e.g., components, architecture) usually have a great variety of features. Test prioritization actually entails the determination of an order relation, or relations, among these features. In this respect, "test prioritization" refers to the comparison of the software objects qualified with different attributes, which is an NP-complete [12] problem to solve in general.

In this paper, test prioritization is performed by means of cluster analysis, which enables adjustment of the priority of test cases in accordance with the importance of the events that they are composed of, even if all the test cases under consideration have equal lengths in terms of the number of events included. In the proposed approach, test cases are first generated from a model that represents the desired behavior of the system, and then a degree of preference is assigned to each generated test case. The degree of preference associated with each test case is then adjusted according to the result of the classification of the component events into clusters based on a set of 13 predefined attributes. Those attributes depend on the features of the SUT and hence they get values proportional to their significance to the model and test sequences, but no prior information about the kind and the number of the failures observed during the previous tests of the SUT is needed to determine those values.

One of the contributions of the study presented in this article is to introduce 13 attributes that enable generating test cases from a model hierarchy with several levels as if it is a single-level model. When there is a single-level model to represent the SUT, prioritization of the complete event sequences (CESs) based on the

events that compose them is straightforward compared to that of a model hierarchy with more than one level. In a model hierarchy, the models at the lower levels will occur in CESs over and over again if we generate test cases from the hierarchy in an ordinary fashion, which results in redundancy and hence arbitrary effects on the priority values to be calculated for CESs based on the constituent events. By means of the 13 attributes that we propose, one can represent a complex SUT as a model hierarchy, instead of a single model, in order to get a manageable representation of the SUT and then prioritize the CESs as if the SUT is represented by a single model. The introduced set of 13 attributes takes into account the empirical fact that the lower the level of a model in the hierarchy is, the lesser is the number of faults to be revealed by the CESs generated from the model [13]: the proposed prioritization approach gives less weight to the events in the CESs generated from the models at lower levels than to those generated from the models at higher levels.

Almost all of the previous approaches in the literature require prior information [14–17], for the reason that they focus on regression testing, where the concern is the revalidation of SUT after it has been modified, i.e. after detected faults are corrected. One of the benefits of the proposed approach is that it can be deployed in any stage of software development to reflect the preferences of the tester wherever he/she needs to select a subset among all possible/available test cases in favor of the total testing cost. Although the authors' previous works [13,18–23] also defined similar attributes for events and assigned values to them in order to determine the degree of preference by means of cluster analysis [24], the approach presented in this article is different in that it classifies the events using two techniques adapted from soft computing, namely neural networks and fuzzy logic [22,23]. Details and the theoretical basis of the aforementioned clustering-based prioritization approaches are given in the authors' previous works [21–23]. A high-level flowchart that shows the execution steps of the considered approaches is given in Figure 1.

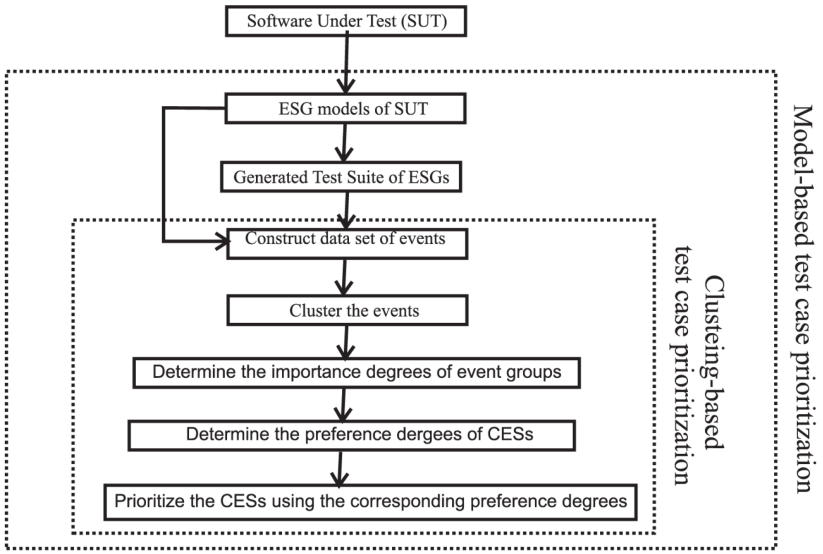

Figure 1. Clustering-based test case prioritization approach.

The proposed approach is evaluated over six case studies on a large, commercial web-based touristic portal. Each case study corresponds to a particular module of the portal, which is modeled using several ESGs related within a hierarchy. Thirteen attributes, which are introduced in this study, are then used to quantify the significance of individual events composing the ESGs in terms of their expected fault-revealing capability. In order to obtain the final prioritized test cases, events are clustered using both neural networks trained by

the adaptive competitive learning (ACL) algorithm and the fuzzy c-means (FCM) algorithm. The Test Suite Designer tool [25] is used for ESG-based modeling and also generating CESs from the obtained model. Each CES corresponds to one test case in the study. To implement the ACL clustering algorithm, a custom software is developed in the Delphi platform. The MATLAB fuzzy tool is used for the application of FCM clustering.

The paper is organized as follows. Section 2 extends and deepens the discussion on related works, which has already been initiated in this introductory section. Section 3 presents the necessary background about the soft computing techniques employed. Section 4, the heart of the paper, presents the proposed approach to the test case prioritization problem. In Section 5, the six case studies of the proposed approach are presented. Section 6 discusses the results of the case studies and lessons learned for practice. Finally, Section 7 gives hints for research work planned and concludes the paper.

2. Related works

Software testing is a time-consuming and expensive process [1]. The results of previous empirical researches have shown that testing can account for almost 50% of the total cost of software development. Real-life systems often require a large number of tests, which cannot be run exhaustively due to time and cost constrains. It is therefore important to decide which part of the software should be tested first. For this purpose, test case selection, test suite reduction, and test case prioritization methods are suggested in the literature [26–35].

The objective of test case selection or test case filtering is to reduce the number of test cases in a test suite, basically by means of partition testing. In partition testing, test cases are divided into equivalence classes and the tests are selected in such a way that at least one test case from each equivalence class is tested [16,17,36]. On the other hand, the objective of test suite reduction is to reduce the number of test cases and eliminate repetitive test cases from the test suite, while maintaining the effectiveness of the original test suite in fault detection [26,27].

The objective of test case prioritization is to rank the test cases according to an adequacy criterion or criteria [28,37]. In other words, the objective is to reveal faults earlier so as to reduce the total cost of testing. There are numerous test case prioritization methods in the literature, but in contrast to the approach presented in this article, they are mostly code-based and they focus on regression testing. It is also different from black-box and other model-based testing approaches in that, as opposed to the proposed approach, they use test case prioritization models that are based on "usage models" and "usage profiles", which might not be available for all existing software.

The test case prioritization problem (TCPP) was originally defined by Rothermel et al. [28] as follows:

Given: A test suite T; $T^{'}$ and T'' are different variations of the test suite; the set PT of permutations of T; a function f from PT to the real numbers, which represents the preference of the tester while testing.

Problem:

$$\text{Find}\, T^{'} \in PT \,\text{such that}\, \left(\forall T^{''}\right)(T'' \neq T')[f(T') \geq f(T'')].$$

In this line of research, Wong et al. [38] were the first researchers to introduce a test case prioritization method with the main focus on reducing cost per additional coverage. Later on, many researchers attacked the TCPP. Srivastava proposed a technique that prioritizes test cases based on the fault detection rate calculated from average fault found per minute [29]. Kim and Porter proposed a test prioritization technique based on historical execution data [30]. They conducted an experiment to assess its effects on the long-run performance of resource-constrained regression testing. Srikanth et al. suggested a value-driven approach to system-level test case

prioritization, called the prioritization of requirements for test, based upon four factors: requirements volatility, customer priority, implementation complexity, and fault proneness of the requirements [31]. Krishnamorthi et al. proposed a prioritization technique at the system level for both new and regression test cases [32]. Srivastava and Thiagarajan built Echelon, a test prioritization system, which prioritizes a given set of tests based on what changes have been made to the software before [33]. Jeffrey and Gupta suggested an approach to prioritize test cases that is based not only on total statement (branch) coverage but also takes into account the number of statements (branches) influencing, or at least having potential to influence, the output produced by the test case [34]. Bryce and Memon proposed a testing technique that extends graphical user interaction (GUI) testing. They prioritized existing test suites for four GUI-based programs by t-way interaction coverage [35].

In the same line of research, Leon et al. compared coverage-based and distribution-based techniques for filtering and test case prioritization [39]. Since distribution-based techniques for filtering and prioritization identify features of the profile distributions that are potentially relevant to revealing faults, these features can be used to guide the selection or prioritization process. They considered two types of distance-based filtering and prioritization techniques: cluster filtering [40] is based on automatic cluster analysis [41], and failure-pursuit sampling is an adaptive extension of cluster filtering that seeks to exploit the observation that tests with failures are often clustered together in small clusters. Their experimental results showed that both coverage-based and distribution-based filtering techniques can exhibit good defect-detection efficiency [40].

On the other hand, Panigrahi et al. proposed a model-based regression test case prioritization technique for object-oriented programs [42]. This technique involves constructing a graph model of the source code to represent the dependencies between control and data, as well as the relations among objects such as inheritance, polymorphism, and message passing. This model is further augmented with information, such as message paths and object states that are available from the UML design models [42]. They used an extended system dependence graph, which was introduced by Horwitz et al. [43] and extended by Larsen et al. [44], for modeling the program and also took into account the various relationships that exist among program elements.

Korel et al. proposed a model-based test reduction technique that uses extended finite state machine (EFSM) model dependence analysis to reduce a given regression test suite. EFSM models are usually depicted as graphs where states represent nodes and transitions represent directed edges between the states. Korel et al. performed an experimental study in order to compare simple code-based and model-based test prioritization methods [45]. The result showed that model-based test prioritization may significantly improve the early fault detection as compared to code-based test prioritization. This method, however, needs prior information of the system (e.g., source code, number of faults), and it could only be used in system retesting. The model-based methods of test prioritization use only the behavioral model of a given system for the purpose of testing. To obtain such a model, several modeling languages have been developed, including EFSM [46], specification description language [47], event flow graphs [48], UML diagrams, ESGs [6,8].

As a measure of how early a prioritized test suite detects faults, Rothermel et al. used a weighted average of the percentage of the faults detected (APFD) [28] during the execution of the test suite. Qu et al. used the normalized version of the APFD to measure the effectiveness of prioritization. Moreover, Elbaum et al. suggested a metric based on the APFD called the new "cost-cognizant" metric [37].

It is worth noting that existing test prioritizations approaches, both code-oriented and model-oriented, apply to regression testing and use code-based criteria, e.g., (code-) coverage and similarity-of-test cases. Bryce et al. applied event-oriented prioritization criteria indirectly to the model, e.g., parameter-value interaction and

measuring of the coverage of windows/actions/call frequency [9]. However, they did not perform any clustering analysis.

To sum up, compared to the above-mentioned related approaches, the main benefit of the suggested test case prioritization technique is that no prior knowledge is needed about the test(s) carried out before, which makes the present technique appropriate for any stage of testing, including regression testing by design. Furthermore, the priority ranking of test cases is determined by means of clustering based on a finite set of predefined attributes.

3. Soft computing

This section introduces the fundamental ideas about soft computing and gives necessary backgrounds for soft computing techniques.

Zadeh was the first researcher to introduce soft computing, as opposed to "hard computing", as a way of building computationally intelligent systems. In the context of hard computing, the major shortcomings are precision, certainty, and rigor. Soft computing, which actually models the human mind, deals with imprecision, uncertainty, partial truth, and approximation to achieve tractability, robustness, and low solution cost [49]. Soft computing methods can therefore be used for finding approximate solutions for real-world problems that contain inaccuracies and uncertainties. Additionally, soft computing techniques do not rely on the assumptions that are common to conventional statistical methods, such as the underlying statistical distribution of data, and thus they are useful in situations where little or no prior knowledge exist [22,23].

The main building blocks of soft computing are neural networks (NNs), fuzzy logic (FL), genetic algorithms (GAs), and probabilistic reasoning (PR). In this study, two popular soft computing techniques, which are based on NNs and FL, are used for test case prioritization.

Soft computing techniques are commonly used for pattern recognition by clustering. Clustering or cluster analysis was first introduced by Tryon in 1939 [50] as an exploratory data analysis tool that aims at assigning different data points into groups (as known clusters) in such a way that the degree of association between two data points is maximal if they belong to the same group and minimal otherwise.

Clustering algorithms can be divided into two groups: hard and fuzzy. Hard clustering based on classical set theory assigns each data point (as an input vector) to exactly one cluster. In fuzzy clustering, a given data point does not necessarily belong to only one cluster; it may have varying degrees of membership to more than one cluster [51,52].

There exist numerous clustering algorithms proposed in the literature [50–56]. They are usually used for processing the input data of complicated classification tasks. In this study, two clustering algorithms are used: one is the ACL algorithm (i.e. hard clustering) [53–55] for unsupervised NNs, and the other is the FCM algorithm (i.e. soft clustering), which is based on FL [56].

4. Model-based test prioritization using cluster analysis

The presented study uses ESGs for several reasons. First, the test process is an event-centric one: events can be externally perceived while states feature internal aspects of the systems modeled. Second, an ESG represents a directed graph and thus results known from graph theory can directly be applied, for example to solve optimization problems related to test coverage. Finally, the ESG representation of the SUT remains unchanged while the code of the SUT will be modified to correct the faults revealed during the test process, provided that the underlying specification is kept valid.

An ESG treats user actions and the corresponding system behavior as events. Tests are then performed by means of a finite sequence of discrete events consisting of the user activities (inputs) as stimulus and the expected system responses (outputs), such that:

(initial) user input $\rightarrow$ *(interim) system response* $\rightarrow$ *user input* $\rightarrow$ *(interim) system response* $\rightarrow \ldots \rightarrow$ *(final) system response.*

Model-based test case prioritization can in this respect be considered as an inexpensive alternative to the existing code-based test prioritization methods, while on the other hand it may be sensitive to the correct/incorrect information provided by the testers/developers. The importance of this method comes to light especially when the source code of the system is unavailable [21–23].

In brief, model-based prioritized testing refers to the use of a model of the SUT for test case generation and prioritization. This study uses ESGs for the purpose of modeling, under the assumption that the behavior of the SUT has correctly been specified and modeled by a set of ESGs. In this respect, to generate test cases from ESGs, arc coverage is used as the criterion. Arc coverage is essentially a measure of to what degree all possible sequences of events (in combination) have been covered. Note that, as a minimal requirement, a test suite should cover all events and event pairs in the ESG by a set of CESs. An optimization problem arises for simultaneously keeping the sum of the lengths of those CESs minimal, which leads to a considerable reduction of the total cost of testing.

In this study, CESs are generated by a variant of the CPP using the Test Suite Designer tool [25] and are then ranked in the decreasing order of their preference degrees, which are quantified indirectly by means of the clustering of the events that compose CESs. For clustering purposes, each event is treated as if it is a point in multidimensional space, with one dimension for each attribute. In this study, we define 13 attributes as given in Table 1. Those attributes (x_{i1} to x_{i13}) are heuristically chosen to qualify an event (i.e. a node in an ESG), primarily to account for an optimized total cost of testing. The attributes are categorized into three subscopes according to the model at hand. If there is a single ESG model with a single level, the first six attributes can be used in relation to the model and the next three in relation to the CESs generated from the model. The last four attributes can be used in the case of an ESG model with multiple levels.

Any user (tester) is encouraged to extend or modify the list in a way that best suits his/her preferences. Eq. (1) is used to determine the values of x_{i9} (only one graph) and x_{i10} (all graphs):

$$Avrf\left(x_i\right)=\frac{1}{d}\left(\sum\nolimits_{q=1}^{r}\frac{f_q\left(x_i\right)}{l\left(CES_q\right)}\right) \tag{1}$$

where $Avrf\left(x_i\right)$ is the averaged frequencies of the usage of event i, r is the number of events composing the qth CES, $f_q(x_i)$ is the frequency of occurrence of event i within CES_q, and $l\left(CES_q\right)$ is the length of CES_q.

From the viewpoint of this study, a cluster is a set of events satisfying the condition that each event is closer (or more similar) to every event (observation) within the associated set than to any event in any other set. Here, the optimal number of clusters, say c, is determined in advance by using the cluster validity algorithm, as described in [57]. Those c clusters are then obtained by means of both ACL and FCM clustering algorithms [21–23].

After obtaining c clusters of events, an importance degree is assigned to every event in each cluster using the importance degree of the cluster. The importance degree of the kth cluster, $ImpD(S_k)$, is obtained by sorting the c clusters in decreasing order of the associated mean vectors $l(\bar{x}_k)$, such that the cluster with the highest $l(\bar{x}_k)$ value has importance degree of 1, implying that it is the most important cluster of events among

Table 1. The attributes list.

Scope	No.	Attribute statements
A single model	x_{i1}	The distance of an event from the start "[" in the CES that contains it.
	x_{i2}	The number of incoming and outgoing edges.
	x_{i3}	The number of nodes that are directly and indirectly reachable from an event, except entry and exit.
	x_{i4}	The number of nodes of a subnode as submenus that can be reached from this node.
	x_{i5}	The balancing degree determines balancing a node as the sum of all incoming edges and outgoing edges for a given node.
	x_{i6}	The number of faulty event pairs connected to the node under consideration (takes the number of all potential faulty events entailed by the event given into account).
Test sequences	x_{i7}	The maximum number of nodes to the entry "[".
	x_{i8}	The total number of occurrences of an event within all CESs, i.e. walks.
	x_{i9}	The averaged frequencies of the usage of events ($Avrf(x_i)$), Eq. (1), within the CESs (only one graph), where $f_q(x_i)$ is frequency of occurrence of event i within CESq and $l(CES_q)$ is the length of CESq. d is determined for events belonging to number of CESs as $d \leq$ r.
Modeling at different levels	x_{i10}	The averaged frequencies of the usage of events ($Avrf(x_i)$), Eq. (1), within the CESs (all graphs).
	x_{i11}	Sequence number of the ESG model.
	x_{i12}	The number of levels in which the ESG model exists.
	x_{i13}	The minimum number of nodes to the entry "[".

all [13,21–24]. The importance index, $imp(x_i)$, of the ith event belonging to the kth cluster is in this respect defined as follows:

$$Imp(x_i) = c - ImpD(S_k) + 1 \tag{2}$$

The preference degree of the qth CES ($PrefD(CES_q)$) can be defined by taking into account the importance index with Eq. (2) of all events belonging to this CES and the frequency of occurrence of event(s) within them. Precisely, the proposed clustering-based prioritization approach ranks CESs according to their preference degrees as given by Eq. (3):

$$PrefD(CES_q) = \sum_{i=1}^{n} Imp(x_i)\mu_{S_k}(x_i) f_q(x_i) \tag{3}$$

where $\mu_{S_k}(x_i)$ is the membership degree of the ith event belonging to the cluster S_k, and $f_q(x_i)$ is the frequency of occurrence of event i within CES_q. It is worth mentioning that, in FCM-based prioritization, $\mu_{S_k}(x_i)$ is in the range of [0,1]; however, in ACL-based prioritization, its value is dichotomous: 1 or 0.

The priorities of the test cases are determined by ranking the calculated $PrefD(CES_q)$ values from highest to lowest. The path with the highest $PrefD(CES_q)$ value has in this respect a priority of 1, which is the highest priority value by design [13,21–23]. The pseudocode of the proposed prioritization algorithm is given in Table 2.

The method based on clustering needs no prior knowledge, such as number of faults, usage profiles, and binary or source code of the SUT, which in fact makes the method radically different from most existing approaches. All of the existing prioritization approaches focus on the test cases in the testing process, but the suggested approaches focus on the importance of the events composing the test cases. The method makes a distinction between the events by clustering. Thus, it reveals differences between the test cases having equal nodes and code coverages, and in this respect it provides an important advantage in terms of test prioritization

and cost reduction. Prioritizing tests cases without needing source codes is also a significant advantage as it protects the confidentiality policies of companies.

Table 2. Clustering-based prioritization algorithm.

Step 1. Construct a set of events $X = \{x_{ij}\}$, where $i = 1, \ldots, n$; $i \epsilon N$ is an event index and $j = 1, \ldots, p$; $j \epsilon N$ is an attribute index.
Step 2. Cluster the events using both ACL and FCM clustering.
Step 3. Classify the events into c crisp groups (using ACL NN-based classification) and fuzzy qualified groups (using FCM-based classification).
Step 4. Determine the importance degrees of groups $ImpD\,(S_k)$ according to length $(\ell(\bar{x}_k))$ of group mean vectors for both types of groups.
Step 5. Determine importance index of event groups (Eq. (2)) with respect to crisp groups and fuzzy qualified groups.
Step 6. An ordering of the CESs as test cases using the corresponding preference degree (PrefD(CES_q)), Eq. (3), for prioritizing the test process in the case of ACL NN-based and FCM-based classification, respectively.

5. Case study: a web-based tourist services marketing system

In this study, the proposed prioritization approach is demonstrated using a web-based tourist services application called ISELTA (Isik's System for Enterprise-Level Web-Centric Tourist Applications). ISELTA was developed by ISIK Touristik Ltd. and the University of Paderborn in cooperation with a commercial enterprise to market various tourist services for traveling, recreation, and vacation. It can be used by hotel owners, travel agents, etc., but also by end-users.

For the purpose of experimental evaluations, the two main modules of ISELTA, the "Specials" and "Prices" modules that are given by screenshots in Figures 2a and 2b, are respectively used. By means of Specials module, one can define, for example, seasonal promotions for specified rooms, and by means of Prices module one can define discounts according to age, number of customers, and number of children. Figure 3 shows the main/top level ESG for the Specials module. The complete model of the Specials module has five sub-ESGs related to the main ESG within the hierarchy. The Specials module is one of the six case studies considered in this article. As opposed to the Specials module, the Prices module has a relatively complex structure (Figure A.1 in the Appendix [on the journal's website] gives the ESG of the module and Table A.1 in the Appendix gives a description of the nodes in this ESG), and so it is divided into five submodules, CH_D, EC, INC1, INC2, and INC3. Each of those five submodules is considered as a separate case study and is also given Figures A.2–A.6 in the Appendix, respectively. Tables A.2–A.6 in the Appendix give descriptions of the nodes in these figures.

The Specials module is modeled using six ESGs in three hierarchical levels and it is tested using the combinations of those ESGs at different levels. In this hierarchy, ESG_1 represents the root; at the second level, ESG_2, ESG_3, ESG_4, and ESG_5 stem from ESG_1; and ESG_6 is at the third level. That is, ESG_2, ESG_3, ESG_4, and ESG_5 are subgraphs of ESG_1, and ESG_6 is a subgraph of ESG_2, ESG_3, ESG_4, and ESG_5. A hierarchy is employed so as to reduce the total number of nodes and edges that has to be considered in testing.

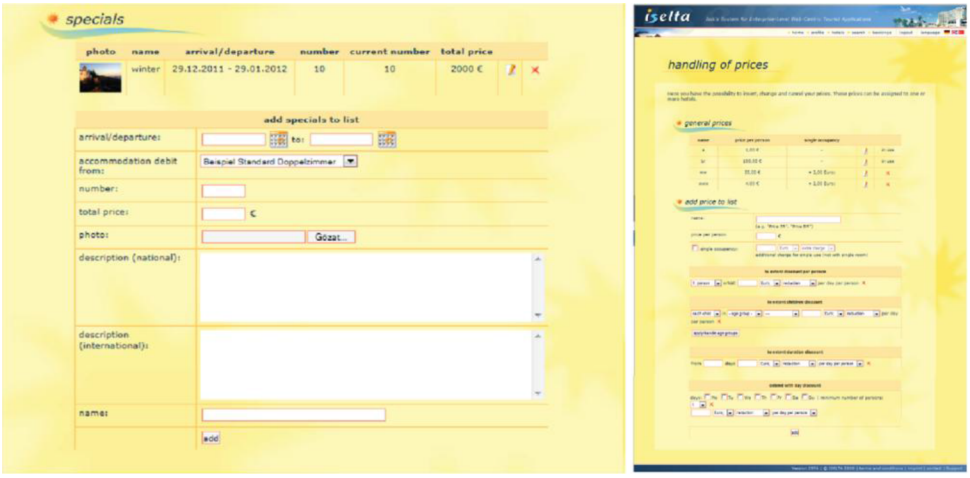

Figure 2. (a) Screenshots of the "Specials" module of ISELTA, (b) screenshots of the "Prices" modules of ISELTA.

In Figure 3, the events S0, S1, and S2+ are used to denote the current condition of the system, where S0 represents no special entry existing in the system, S1 represents the existence of only one special entry, and S2+ represents that there are more than one specials defined. These nodes are used only to distinguish the contexts in modeling, not for testing purposes. Description of the nodes of Specials module is given in Table A.7.

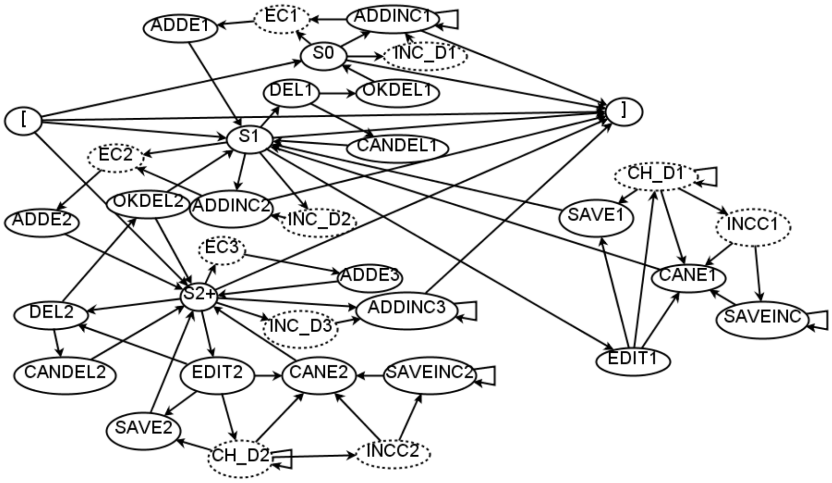

Figure 3. ESG of Specials module representing the first level in the hierarchy (ESG_1).

The nodes that are modeled separately in the second level, INC_D1, INC_D2, INC_D3, EC1, EC2, EC3, CH_D1, CH_D2, CH_D2, INCC1, and INCC2, are indicated by surrounding them with a dashed line. Note that INC_D1, INC_D2, and INC_D3 are modeled using the same graph, ESG_2, given in Figure A.7a. ESG_2 shows how to enter incomplete data in the system. ESG_3, ESG_4, and ESG_5 depict entering complete data, deleting data, and changing data, respectively, as shown in Figures A.7b–A.7d. In a similar fashion, the nodes in ESG_2, ESG_3, and ESG_5, such as ADDDATE1, ADDDATE2, ADDDATE3, D_ENTER, and C_DATE, are detailed in the third level as shown in Figure A.7e. The nodes are indexed so that a single node can be used in the same model but under different conditions or contexts.

The Prices module of ISELTA is divided into five submodules for the sake of clarity in modeling as given in Figure A.1. These modules: are 1) Changing Data (CH_D), 2) Entering Correct Data (EC), 3) Entering

Incorrect Data (INC1) when there is no entered datum at all, 4) Entering Incorrect Data (INC2) when there is an entered datum, and 5) Entering Incorrect Data (INC3) when there is more than one datum entered before. Figures A.8a, A.8b and A.8c show the ESGs of CH_D, EC, and INC3, respectively. In addition, all submodules of Prices and their hierarchies are given from Figure A.2 to A.6 in the Appendix.

The complete model of CH_D has 15 sub-ESGs in four hierarchical layers. The ESG of the CH_D module given in Figure A.8a is at the top level in the hierarchy. Nodes SOC, DISC1, DISC2, DISC3, and DISC4 of CH_D have detailed ESGs at the second level. The complete model of the EC module has 7 sub-ESGs in three hierarchical levels. The ESG of EC module in Figure A.8b represents the root. The nodes SOCDATAE, DISDATAE1, DISDATAE2, DISDATAE3, and DISDATAE4 of EC have detailed models at the second level. The complete model of INC3 given in Figure A.8c also has 7 sub-ESGs in three hierarchical levels, where nodes ISODE3, IDISDE31, IDISDE32, IDISDE33, and IDISDE34 have detailed models at the second level. Similar to the INC3 module, both of the complete models of the INC1 and INC2 modules have 7 sub-ESGs in three hierarchical layers

In summary, the Specials and Prices modules of the ISELTA portal are considered in this study as if they are six "main" modules: 1) Specials, 2) CH_D, 3) EC, 4) INC1, 5) INC2, and 6) INC3, each of which represents a single case study. The details of the six modules are given in the Appendix (from Figure A.1 to A.6). Totally, 351 CESs and 675 events are generated from all the ESGs modeled, and those 351 CESs are capable of revealing a total of 470 faults (Tables A.8 and A.9 in the Appendix). Table A.10 shows the distribution of the total 470 faults on individual modules. The test suites generated for the case studies are given Tables A.11–A.16 in the Appendix.

6. Experimental results and the discussions

In the current practice of testing, it is generally agreed that the sequences having the greatest arc coverage and node coverage should be executed first. However, this also means higher costs because such sequences are generally longer. Thus, choosing the relatively shorter test sequences is also an important requirement. At this point, fault detection performance plays an important role in keeping the whole testing process in balance with respect to the coverage and the cost. Unfortunately, the existing assessment criteria do not always give adequate information on the fault detection performance. Most importantly, they may well be insufficient to make a clear distinction between equal-coverage test sequences, i.e. CESs with equal numbers of events.

Fault detection performance is in this respect an important criterion for the comparison of the test prioritization methods. However, since test engineers usually correct the faults where they find them in the sequential flow of the test process, the observed performance would in general fluctuate, test process from test process, depending on the order of the test cases executed. It is true that a corrected fault will not occur again in the following test sequences, but it is also true that when a fault is detected, the execution of the test sequence cannot continue; it is necessary to rerun the sequence after correcting the fault. This means that the number of faults revealed by an individual test case may change if test cases are executed in a different order. Nevertheless, note that, although the number of faults in a system cannot be known in advance, one can inject specific faults to evaluate the fault detection performance of a given testing approach. In this study, faults are injected deliberately to the systems with mutation testing by means of deleting a line or statement, or changing operators in the code of the system [58]. Faults are not a part of the prioritization process. They are injected only to evaluate the performance of the proposed prioritization techniques.

In this study, 12 test case prioritization techniques are considered for experimental evaluations. Those

techniques can be classified into three groups as given in Table A.17. The first group is composed of the two model-based test case prioritization approaches proposed in this study, where "TCP1" stands for the approach based on ACL and "TCP2" stands for the approach based on FCM. The second group is composed of six well-known code-based test case prioritization approaches (from "TCP3" to "TCP8") in the literature [15]. When the source code of the SUT is available, we can determine, for any test case (CES) given, the number of statements covered, the number of lines of code (LOC) covered, and the number of functions that were exercised by the test case, and we can utilize this information for test case prioritization. "TCP3", "TCP4", and "TCP5" represent the approaches that use the number of LOC covered, the number of functions covered, and the number of statements covered, respectively. The remaining code-based approaches, "TCP6", "TCP7", and "TCP8", are, in one sense, the complements of "TCP3", "TCP4", and "TCP5", respectively: "TCP6" represents the approach that uses the total lines of code not yet covered, "TCP7" the number of functions not yet covered, and "TCP8" the number of statements not yet covered. One can also use additional statement coverage, additional function coverage, and additional LOC coverage [15] for the same purpose. The third group in Table A.17 is a control group with four members, from "TCP9" to "TCP12", each of which serves two purposes: 1) performance evaluation criterion (defined in Section 6.1) and 2) "ideal" test case prioritization approach for the performance evaluation criterion itself.

For the CESs generated for the Specials module, the priority values that are yielded from the considered approaches, TCP1 through TCP11, are listed in Table A.18. Priority values assigned to the CESs generated from the INC3 module are also given in Table A.19 in the Appendix.

6.1. Performance evaluation criteria: control group

In this study, four new performance evaluation criteria, which expose the fault detection success of test sequences, are used for the evaluation of the proposed clustering-based test prioritization methods. During the testing, the events that capture the faults are determined and then the following evaluation criteria are used to derive different ideal orderings and compare the employed strategies.

- Number of Events Capturing Unique Faults (TCP9): For this criterion, the ideal CES ordering is obtained by sorting the generated CESs in decreasing order of the number of the events in each CES that can capture unique faults. Note that there are events having different labels but essentially the same functionality, such as TODAY11, TODAY12, TODAY22, and TODAY21, where the difference in labeling indicates that the events that they follow (i.e. the triggering events) are different. Actually, the CESs that contain one of the events TODAY11, TODAY12, TODAY22, and TODAY21 could reveal the same fault(s) depending on the execution order of the CESs. Since each of these four events is able to catch the same unique fault(s), they are considered as unique fault catcher events in the evaluation of the proposed methods.

- Number of Events Capturing Repetitive Faults (TCP10): For this criterion, the ideal CES ordering is obtained by sorting the generated CESs in decreasing order of the number of the occurrences of each event that can capture faults, not necessarily the unique ones. In the context of the TCP10 criterion, it is assumed that the faults detected during the execution of any CES are left as are (i.e. they are not corrected, so they are repetitive across different CESs containing the same fault catching events). Moreover, in contrast to the TCP9 criterion, all the occurrences of the events are considered as "fault catcher" events, in order to take into account the empirical fact that the same event could reveal different faults when it is triggered by different events. This means, as a result, that the TCP10 values of individual

CESs are calculated independently from each other.

- Number of Events Capturing Nonrepetitive Faults (TCP11): Assuming that each CES is executed by starting the SUT from scratch (i.e. it is returned back to the original state after each CES execution, so that the CES executions can be considered mutually independent), the number of faults revealed by each CES can be calculated as if it is executed as the first one. Sorting the generated CESs in decreasing order of the numbers of faults that are revealed in this way then gives the ideal ordering. Basically, the TCP11 value of a particular method is given by the number of faults that the method itself could detect if it is considered separately. The main difference between TCP10 and TCP11 is that there the revealed faults are corrected while the CESs yielded from each method are executed. Without losing generality, in the context of TCP11, it is assumed that faults are nonrepetitive in the sense that a particular fault is detected and corrected. Additionally, the same fault will not occur again during the execution of the test suite under evaluation.
- Number of CESs Required to Reveal All Faults (TCP12): This criterion is different from the former ones in that it is simply given by the number of CESs required to reveal all faults in the SUT, when the CESs are executed in the order determined by the prioritization method in use.

6.2. Results of the evaluation of the suggested prioritization approaches

Based on the three evaluation criteria TCP9, TCP10, and TCP11, the results of the evaluation of the considered test case prioritization approaches, TCP1 through TCP8, on the Specials module are depicted in Figures A.9, A.10 and A.11 respectively. In Figures A.9 through A.11, the horizontal axis shows the rank value or the priority value (i.e. the execution order) assigned to the CESs by each prioritization approach, while the vertical axis shows the cumulative total of the observed values of the TCP9, TCP10, and TCP11 criteria, respectively.

As seen in Figure A.9, the considered test case prioritization approaches tend to accumulate in two groups based on the criterion TCP9: one is a mixture of model-based and code-based approaches including TCP1, TCP2, TCP3, TCP4, and TCP5, and the other is composed of the remaining code-based approaches, TCP6, TCP7, and TCP8. In contrast to code-based approaches TCP6, TCP7, and TCP8, model-based approaches TCP1 and TCP2 and code-based approaches TCP3, TCP4, and TCP5 show a performance close to the performance of the ideal approach, TCP9. One of the reasons for this is that the code-based approaches TCP6, TCP7, and TCP8 give high precedence to short test cases, relative to long test cases. When short test cases are executed first, although the cost increases gradually from start to end with steps as small as possible, it appears that this results in a considerable amount of loss in capturing unique faults at early stages compared to that of the ideal.

As seen in Figures A.10 and A.11, the same situation occurs when the approaches are evaluated based on criteria TCP10 and TCP11. The analyses of some modules, having different results, are given in Figures A.12–A.17 in the Appendix. The evaluation of the INC1 and INC2 modules yields similar results with CH_D module.

To test whether there are significant differences between the test case prioritization approaches, TCP1 through TCP8, and the control group approaches, TCP9 through TCP11, one can use the well-known nonparametric hypothesis testing method, called the Friedman method in statistics. The Friedman test [59] can be used to compare three or more related samples. It is in general identical to the balanced two-way analysis of variance (ANOVA) [60], but in particular it is different from ANOVA in that it tests only for row effects after adjusting for possible column effects, and most importantly it makes no assumption about the underlying distribution of

the data. In our case, the Friedman test is used for testing the null hypothesis, H_0, against the alternative, H_A, as given by:

H_0 : *There is no significant difference between the test case prioritization approaches.*

H_A : *There exists a significance difference.*

The nonparametric Friedman test compares the sample medians of several groups (in our case, test case prioritization approaches) to test the null hypothesis (H_0) that they are all the same against the alternative (H_A) that they are not all the same. The P-value that the Friedman test returns is used to cast doubt on the null hypothesis. A sufficiently small P-value indicates that at least one approach is significantly different in sample median than the others. To determine whether a result is "statistically significant", a critical P-value is chosen by the researcher, generally agreed to be 0.05.

Table A.20 shows the results of the Friedman tests carried out to decide whether there is a significant difference among prioritization approaches TCP1 through TCP8 and TCP9, among TCP1 through TCP8 and TCP10, and among TCP1 through TCP8 and TCP11. As seen, all the observed P-values are less than 0.05. Thus, the null hypothesis can be rejected with 95% confidence for each of the three tests. That is, there is enough evidence in the data at hand to conclude, with 95% confidence, that at least one approach is significantly different in ranking CESs from the others.

As a matter of fact, the information that at least one approach is significantly different in ranking CESs from the others is too general. In practice, we usually need information about which pairs of approaches are significantly different from each other and which are not. A test that can provide such information is called a multiple comparison test in statistics. The multiple comparisons of the considered prioritization approaches are given in Figures A.18a–A.18c. They use the Tukey honestly significant difference criterion (Tukey HSD) [61], which is based on the Studentized range distribution.

In Figures A.18a–A.18c, the sample median (i.e. mean rank) associated with each prioritization approach is marked by a circle. Horizontal solid lines crossing the circles represent 95% confidence intervals (CIs) for the associated prioritization approaches. The prioritization approaches whose 95% CIs do not overlap are those approaches that are significantly different from each other in ranking CESs. The vertical dashed lines emphasize the end points of the 95% CIs associated with the control group approaches TCP9, TCP10, and TCP11.

As shown in Figure A.18a, except for the code-based prioritization approaches TCP6, TCP7, and TCP8, model-based approaches TCP1 and TCP2 and code-based approaches TCP3, TCP4, and TCP5 are not significantly different in capturing unique faults from each other, though they are different from the ideal approach, TCP9. For the Specials module, the fact that the results obtained for criteria TCP10 (Figure A.18b) and TCP11 (Figure A.18c) are the same as the result obtained for criterion TCP9 suggests that the proposed model-based approaches are the baseline alternatives for code-based approaches. This is the case if the source code of the SUT is available to the tester. In the case that the source code of the SUT is not available, this statistical analysis suggests that the proposed model-based approaches can be used with 95% confidence.

Except for module INC3, the results of the statistical analyses performed for all the modules are similar to each other (presented in the Appendix: Figures A.19–A.22). For the case of INC3, the results of the Friedman tests performed show that the observed discrepancy between the model-based approaches and the code-based approaches is statistically significant at a significance level of 0.05. Table A.21 lists the corresponding Friedman ANOVA tables. Tables A.22–A.25 show the results of Friedman ANOVA tests for the CH_D, EC, INC1, and INC2 modules, respectively.

As seen in Figure A.23, the model-based approaches show performances in revealing faults not significantly

different from that of ideals TCP9, TCP10, and TCP11, while in contrast the differences between the code-based approaches and the ideals are statistically significant. In particular, Figures A.23a–A.23c illustrate Pareto-based graphs for the INC3 module, and Figures A.23d–A.23f illustrate the results of multiple comparisons for the module according to TCP9, TCP10, and TCP11, respectively.

The CESs obtained from the INC3 module are relatively short in number of events compared to the CESs obtained from the other modules. This makes the CES rankings yielded from approaches TCP3, TCP4, and TCP5 close to the CES rankings yielded from approaches TCP6, TCP7, and TCP8. Thus, the CES rankings yielded from all the code-based approaches differ from the CES rankings yielded from the model-based approaches, and most importantly they differ as a group from the ideal CES rankings yielded from TCP9, TCP10, and TCP11, whereas the model-based approaches yield CES rankings close to the ideal. The case of INC3 exemplifies an important advantage of model-based test case prioritization over code-based prioritization, which occurs when CESs have equal lengths in number of events. Note that code-based approaches would in general assign the same priority value to the CESs with equal lengths, as opposed to model-based approaches.

The total cost of correction of all faults (TCP12) is calculated based on both the number of CESs required to reveal all the faults and the coverage of them in terms of events. Here, criterion TCP12 represents the total cost of the whole fault detection process by fixing the cost of starting the execution of each CES from α and the cost of executing each event to β, where α and β are determined considering the test budget. As shown in Table A.26, the total cost of revealing all faults by means of model-based prioritization approaches is in general lower than that of the code-based prioritization approaches.

6.3. Limitations

The primary limitation of the proposed prioritization approaches is that it can be affected by changes in the model, concerning the generated test sequences. In addition, only behavioral sequence-based faults are revealed and corrected; the logical and/or calculation errors, which usually depend on the examination of outputs, are ignored. Finally, the performance evaluation criteria require data of an experimental type. Therefore, the derivation of the ideal rankings for different criteria and the results of the comparisons with the ideal rankings are all dependent on the actual test execution data. This means that the real performance of the suggested prioritization methods can only be observed at the end of the test process.

7. Conclusion

The model-based, coverage- and-specification-oriented approach proposed in this paper provides an effective algorithm for ordering a given set of test cases with respect to their degree of preference as perceived by the tester, which results in a set of priority-sorted test cases. The degree of preference associated with a test case is determined by means of the clustering of the events based on the 13 attributes defined in this study. The approach classifies the events (nodes of ESG) by means of cluster analysis based on both ACL and FCM algorithms. The proposed approach needs no prior knowledge, such as numbers of faults or binary or source code of the SUT, which in fact makes the proposed approach radically different from the code-based approaches.

The results of the experimental analyses performed using a commercial application show that the fault detection performance of the proposed model-based approaches is in general not significantly different from that of the code-based approaches considered in this study. Together with the fact that model-based approaches need no source code, and so they protect the confidentiality policies of companies, we concluded that the proposed model-based test case prioritization approaches are capable to be the baseline alternatives over the code-based approaches in general.

At present, the proposed 13 attributes have equal weights. One of the future works is to use multivariate data analysis techniques in order to get better weighting schemes, such as principal component analysis and factor analysis. Another study planned includes application of the proposed approaches to a broader class of testing problems, e.g., to multiple-metrics-based testing where a family of software measures is used to generate tests. As mentioned in Section 2, some recent work applied event-oriented prioritization criteria indirectly to the model, e.g., parameter-value interaction or measuring the coverage of windows/actions/call frequency [9]. As our approach is also event-oriented [8], we see some chances that in our future work we can attempt to extend, and maybe uniformly structure, our attributes' catalogue, taking this aspect into account.

References

[1] B. Beizer, Software Testing Techniques, 2nd ed., New York, Van Nostrand Reinhold, 1990.

[2] R.V. Binder, Testing Object-Oriented Systems: Models, Patterns and Tools, Boston, Addison-Wesley, 2000.

[3] A.P. Mathur, Foundations of Software Testing, New Delhi, Addison-Wesley Professional, 2008.

[4] J.B. Goodenough, S.L. Gerhart, "Toward a theory of test data selection", IEEE Transactions on Software Engineering, Vol. 1, pp. 156–173, 1975.

[5] M.P.E. Heimdahl, D. George, R. Weber, "Specification test coverage adequacy criteria = specification test generation inadequacy criteria", in: Proceedings of HASE'04, Tampa, FL, USA, pp. 178–186, 2004.

[6] F. Belli, "Finite-state testing and analysis of graphical user interfaces", in: Proceedings of ISSRE'01, Hong Kong, pp. 34–43, 2001.

[7] F. Belli, C.J. Budnik, L. White, "Event-based modeling, analysis and testing of user interactions – approach and case study", Software Testing, Verification & Reliability, Vol. 16, pp. 3–32, 2006.

[8] F. Belli, M. Beyazit, A. Memon, "Testing is an event-centric activity", Proceedings of the 6th IEEE International Conference on Software Security and Reliability, SERE-C, pp. 198–206, 2012.

[9] C. Bryce, S. Sampath, A.M. Memon, "Developing a single model and test prioritization strategies for event-driven software", IEEE Transactions on Software Engineering, Vol. 37, pp. 48–64, 2011.

[10] A.M. Memon, M.E. Pollack, M.L. Soffa, "Hierarchical GUI test case generation using automated planning", IEEE Transactions on Software Engineering, Vol. 27, pp. 144–155, 2001.

[11] J. Edmonds, E.L. Johnson, "Matching, Euler tours and the Chinese Postman", Mathematical Programming, Vol. 5, pp. 88–124, 1973.

[12] S.A. Cook, "The complexity of theorem-proving procedures", in: Proceedings of STOC'71, New York, pp. 151–158, 1971.

[13] F. Belli, N. Gökçe, "Test prioritization at different modeling levels", Communications in Computer and Information Science, Vol. 117, pp. 130–140, 2010.

[14] J.M. Kim, A. Porter, "A history-based test prioritization technique for regression testing in resource constrained environments", in: Proceedings of ICSE 2002, Orlando, FL, USA, pp. 119–129, 2002.

[15] S. Elbaum, A. Malishevsky, G. Rothermel, "Test case prioritization: a family of empirical studies", IEEE Transactions on Software Engineering, Vol. 28, pp. 182–191, 2002.

[16] Y.F. Chen, D.S. Rosenblum, K.P. Vo, "Test Tube: a system for selective regression testing", in: Proceedings of ICSE'94, Sorrento, Italy, pp. 211–222, 1994.

[17] G. Rothermel, M.J. Harrold, "A safe, efficient algorithm for regression test selection", in: Proceedings of ICSM'93, Montreal, pp. 358–367, 1993.

[18] N. Gökçe, M. Eminov, F. Belli, "Coverage-based, prioritized testing using neural network clustering", Lecture Notes in Computer Science, Vol. 4263, pp. 1060–1071, 2006.

[19] F. Belli, M. Eminov, N. Gökçe, "Prioritizing coverage-oriented testing process-an adaptive-learning-based approach and case study", 31st Annual International Computer Software and Applications Conference, COMPSAC 2007, Vol. 2. pp. 197–203, 2007.

[20] F. Belli, M. Eminov, N. Gökçe, "Coverage-oriented, prioritized testing – a fuzzy clustering approach and case study", Lecture Notes in Computer Science, Vol. 4746, pp. 95–110, 2007.

[21] F. Belli, M. Eminov, N. Gökce, "Model-based test prioritizing - a comparative soft computing approach and case studies", Lecture Notes in Artificial Intelligence, Vol. 5803, pp. 427–434, 2009.

[22] F. Belli, M. Eminov, N. Gökçe, W.E. Wong, "Prioritizing coverage-oriented testing process - an adaptive-learning-based approach and case study", Series on Software Engineering and Knowledge Engineering, Vol. 20, pp. 1–22, 2011.

[23] N. Gökçe, Determination of Model Based Test Priorities By Clustering Approach", PhD, Muğla Sıtkı Koçman University, Muğla, Turkey, 2012 (in Turkish).

[24] A. Jain, M. Murty, P. Flynn, "Data clustering: a review", ACM Computing Surveys, Vol. 31, pp. 264–323, 1999.

[25] Angewandte Datentechnik, Test Suite Designer Tool, Paderborn, Germany, University of Paderborn.

[26] W.E. Wong, J.R. Horgan, S. London, A.P. Mathur, "Effect of test set minimization on fault detection effectiveness", in: Proceedings of ICSE'95, Seattle, pp. 41–50, 1995.

[27] G. Rothermel, M.J. Harrold, J.V. Ronne, C. Hong, "Empirical studies of test-suite reduction", Software Testing, Verification & Reliability, Vol. 12, pp. 219–249, 2002.

[28] G. Rothermel, R.H. Untch, C. Chu, M.J. Harrold, "Prioritizing test cases for regression testing", IEEE Transactions on Software Engineering, Vol. 27, pp. 929–948, 2001.

[29] P.R. Srivastava, "Test case prioritization", Journal of Theoretical and Applied Information Technology, Vol. 2005, pp. 178–181, 2005.

[30] J.M. Kim, A. Porter, "A history-based test prioritization technique for regression testing in resource constrained environments", in: Proceedings of ICSE'02, Orlando, FL, USA, pp. 119–129, 2002.

[31] H. Srikanth, L. Williams, J. Osborne, "System test case prioritization of new and regression tests", in: Proceedings of ISESE 2005, Noosa Heads, Australia, pp. 1–10, 2005.

[32] R. Krishnamoorthi, S.A. Sahaaya Arul Mary, "Requirement based system test case prioritization of new and regression test cases", International Journal of Software Engineering and Knowledge Engineering, Vol. 19, pp. 453–475, 2009.

[33] A. Srivastava, J. Thiagrajan, "Effectively prioritizing test in development environment", in: Proceedings of ISSTA2002, Rome, pp. 97–106, 2002.

[34] D. Jeffrey, N. Gupta, "Test case prioritization using relevant slices", in: Proceedings of COMPSAC'06, Chicago, Vol. 1, pp. 411–420, 2006.

[35] C. Bryce, A.M. Memon, "Test suite prioritization by interacting coverage", in: Proceedings of DoSTA2007, Dubrovnik, Croatia, pp. 1–7, 2007.

[36] A. Ensan, E. Bagheri, M. Asadi, D. Gasevic, Y. Biletskiy, "Goal-oriented test case selection and prioritization for product line feature models", in: Proceedings of INTG'11, Las Vegas, pp. 291–298, 2011.

[37] S. Elbaum, A. Malishevsky, G. Rothermel, "Incorporating varying test costs and fault severities into test case prioritization", in: Proceedings of ICSE-01, Toronto, pp. 329–338, 2001.

[38] W. Wong, J. Horgan, S. London, H. Agrawal, "A study of effective regression testing in practice", in: Proceedings of ISSRE 1997, Albuquerque, NM, USA, pp. 230–238, 1997.

[39] D. Leon, A. Podgurski, "A comparison of coverage-based and distribution-based techniques for filtering and prioritizing test cases", in: Proceedings of ISSRE 2003, Denver, CO, USA, pp. 442–453, 2003.

[40] W. Dickinson, D. Leon, A. Podgurski, "Finding failures by cluster analysis of execution profiles", in: Proceedings of ICSE2001, Toronto, pp. 339–348, 2001.

[41] A.K. Jain, R.C. Dubes, Algorithms for Clustering Data, Upper Saddle River, NJ, USA, Prentice Hall, 1988.

[42] C.R. Panigrahi, R. Mall, "Model-based regression test case prioritization", in: Proceedings of ICISTM 2010, Bangkok, pp. 380–385, 2010.

[43] S. Horwitz, T. Reps, D. Binkley, "Interprocedural slicing using dependence graphs", ACM Transactions on Programming Languages and Systems, Vol. 12, pp. 26–61, 1990.

[44] L. Larsen, M. Harrold, "Slicing object-oriented software", in Proceedings of ICSE'96, Berlin, pp. 495–505, 1996.

[45] B. Korel, G. Koutsogiannakis, "Experimental comparison of code-based and model-based test prioritization", in: Proceedings of ICSTW'09, Denver, CO, USA, pp. 77–84, 2009.

[46] K. Cheng, A. Krishnakumar, "Automatic functional test generation using the extended finite state machine model", in: Proceedings of DAC'93, Dallas, TX, USA, pp. 86–91, 1993.

[47] R. Dssouli, K. Saleh, E. Aboulhamid, A. En-Nouaary, C. Bourhfir, "Test development for communication protocols: towards automation", Computer Networks, Vol. 31, pp. 1835–1872, 1999.

[48] A.M. Memon, "An event-flow model of GUI-based applications for testing", Software Testing, Verification & Reliability, Vol. 17, pp. 137–157, 2007.

[49] L.A. Zadeh, "Fuzzy logic, neural networks, and soft computing", Communications of the ACM, Vol. 37, pp. 77–84, 1994.

[50] R.C. Tryon, Cluster Analysis, New York, McGraw-Hill, 1939.

[51] J.C. Dunn, "A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters", Journal of Cybernetics, Vol. 3, pp. 32–57, 1973.

[52] J.C. Bezdek, Pattern Recognition with Fuzzy Objective Function Algorithms, New York, Plenum Press, 1981.

[53] K. Fukushima, "Cognitron: a self-organizing multilayered neural network", Biological Cybernetics, Vol. 20, pp. 121–136, 1975.

[54] S. Grossberg, "Adaptive pattern classification and universal recoding: Part I: Parallel development and coding of neural feature detectors", Biological Cybernetics, Vol. 23, pp. 121–134, 1976.

[55] D.E. Rummelhart, D. Zipser, "Feature discovery by competitive learning", Journal of Cognitive Science, Vol. 9, pp. 75–112, 1985.

[56] F. Hoppner, F. Klawonn, R. Kruse, T. Runkler, Fuzzy Cluster Analysis, New York, John Wiley, 1999.

[57] D.J. Kim, Y.W. Park, D.J. Park, "A novel validity index for determination of the optimal number of clusters", IEICE Transactions on Information and Systems, Vol. D-E84, pp. 281–285, 2001.

[58] R.A. DeMillo, R. J. Lipton, F.G. Sayward, "Hints on test data selection: help for the practicing programmer", Computer, Vol. 11, pp. 34–41, 1978.

[59] M. Friedman, "The use of ranks to avoid the assumption of normality implicit in the analysis of variance", Journal of the American Statistical Association, Vol. 32, pp. 675–701, 1939.

[60] H. Scheffé, The Analysis of Variance, New York, Wiley, 1959.

[61] Y. Hochberg, A.C. Tamhane, Multiple Comparison Procedures, Hoboken, NJ, USA, John Wiley & Sons, 1987.