2016 IEEE Ninth International Conference on Software Testing, Verification and Validation Workshops

Dynamic Integration Test Selection Based on Test Case Dependencies

Sahar Tahvili*†, Mehrdad Saadatmand*, Stig Larsson*, Wasif Afzal†, Markus Bohlin*, Daniel Sundmark†

*SICS Swedish ICT Västerås AB, Sweden
†Mälardalen University, Västerås, Sweden
Email: {sahart, mehrdad, stig.larsson, markus.bohlin}@sics.se
† {wasif.afzal, daniel.sundmark}@mdh.se

***Abstract*—Prioritization, selection and minimization of test cases are well-known problems in software testing. Test case prioritization deals with the problem of ordering an existing set of test cases, typically with respect to the estimated likelihood of detecting faults. Test case selection addresses the problem of selecting a subset of an existing set of test cases, typically by discarding test cases that do not add any value in improving the quality of the software under test. Most existing approaches for test case prioritization and selection suffer from one or several drawbacks. For example, they to a large extent utilize static analysis of code for that purpose, making them unfit for higher levels of testing such as integration testing. Moreover, they do not exploit the possibility of dynamically changing the prioritization or selection of test cases based on the execution results of prior test cases. Such dynamic analysis allows for discarding test cases that do not need to be executed and are thus redundant. This paper proposes a generic method for prioritization and selection of test cases in integration testing that addresses the above issues. We also present the results of an industrial case study where initial evidence suggests the potential usefulness of our approach in testing a safety-critical train control management subsystem.**

***Keywords*—Software testing, Integration testing, Test selection, Test prioritization, Fuzzy, AHP, Optimization**

I. Introduction

While different characteristics of test cases can be evaluated in an offline fashion to determine and select which test cases to execute, the verdict of a test case can also serve as another factor in selection of other test cases to execute [1], [2]. Since the complexity of integration testing increases as the number of subsystems grows [3], considering the dependency between test cases plays a critical role for efficient use of test execution resources. This paper introduces a generic approach for combined static and dynamic prioritization and selection of test cases for integration testing. The prioritization is based on the *dependency degree* of each test case. Further prioritization is performed among test cases at each *dependency degree* level using the Fuzzy Analytic Hierarchy Process technique (FAHP, see [4]); a structured method where properties are expressed using degrees of truth. The approach is close to the way people usually reason, and therefore suitable to this type of complex decision problem. As a prerequisite, we assume the existence of a directed dependency relation, capturing information on which components use other components. In industry, such dependencies between test cases are usually found using reverse engineering [5], but also source code analysis [6], interviews with experts and analysis of documentation may be useful. In the setting of test-driven integration testing, it is often the case that test cases exist for components which have not been implemented yet, making interviews and documentation the most practical source of this type of information.

In detail, the proposed approach consists of the following two phases (offline and online) and four steps in total:

1) (Offline) The test cases are partially ordered by calculating a *dependency degree* for each test case. The *dependency degree* of a test case indicates the extent to which the execution of a test case is redundant given that another test case fails. As a result of this step, some test cases may end up having the same *dependency degree*.
2) (Offline) Test cases with the same *dependency degree* are then prioritized by applying FAHP, producing an ordered set of test cases at each *dependency degree*.
3) (Online) During test execution, test cases are then selected one by one from each ordered set and in ascending order of *dependency degrees*. In this phase, when a test case fails, the test cases that are dependent on it are evaluated to determine if those dependent cases will also fail due to the failure of the former or not, hence avoiding redundancy in test execution.

The overarching objective of the proposed approach is thus to avoid the execution of redundant test cases as well as to prioritize executable test cases based on dependencies and various prioritization criteria, in order to enable more efficient use of testing resources at integration testing.

II. Background and Preliminaries

Selecting a set of core test cases for execution to see whether further testing would be meaningful is beneficial for efficient use of testing resources [3], [7]. In this context, initially a set of test cases can be selected whose results (pass or fail) provide relevant information on which test cases to select next. This can be done by testing first the core features of the system whose failure can result in the failure of other features. In fact,

978-1-5090-3674-5/16 $31.00 © 2016 IEEE
DOI 10.1109/ICSTW.2016.14

by identifying the test cases that will fail because of the failure of some other test cases (result dependency) and avoiding to execute the former when the latter have failed, a better use of testing resources can be achieved. On the contrary, if test cases are selected without considering such dependencies, a test case might fail not because the feature it tests is actually faulty, but that another feature on which it depends on has failed. From this perspective, a dependency chain among test cases can be established. In short, dependencies among test cases can be determined before their execution (offline). The result of each test case during execution, when combined with the dependency information, enables us to dynamically identify which test case to execute next. The idea of using dependency information in identifying redundant test cases is also evaluated and confirmed by Arlt et al. [8] where dependency relationships are derived and inferred from a structured requirements specification.

A. Motivating Example

Many embedded control systems have a possibility to download applications, updates, and configurations making it possible to adapt the behavior of the system to the specific task it will control. This means that it needs to be possible to download the application to the control system and ensure that the integrity of the application is maintained. Different mechanisms, such as using checksums, can be used to confirm that the download is correct. When testing the download function, it is necessary to have a communication channel available with the download functionality implemented. In our example, we have three different communication channels as a part of the system: Bluetooth, Wi-Fi and USB. To be able to test the application download function, at least one of these channels need to pass basic communication tests; hence the dependency between the download function and communication. This is shown in Figure 1. In this case we thus have an OR situation where it is enough that the tests for one of the channels pass before it is useful to test the application download function.

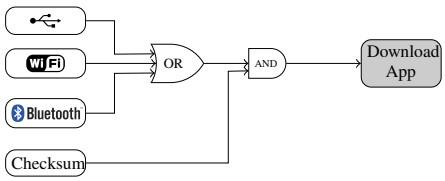


Fig. 1: Dependency with AND-OR relations

We can also get an AND situation for this case. If the tests for creating a checksum fails, there is no point in trying the application download function even if one of the tests for the communication channels passes.

B. Main definitions

To understand the concept of test case dependency in this work, the key terms that are used to describe dependency relationships between test cases are defined below:

Definition 1: **Dependent test case** - Given two test cases A and B, B is *dependent* on A if from the failure of A it can be inferred that B will also fail (result dependency: fail based on fail).

Consequently, based on the result of A, we can decide to also execute B or not. It only makes sense to execute test case B when A has passed. Otherwise, if A has failed and B is also executed, execution of B will not be an optimal use of testing resources, since we know based on the dependency relation and fail result of A that B also fails. As a side note, if test case B can still pass even if A has failed, based on our definition of dependency, B is not (result-) dependent on A. Moreover, it is important to remember that if A passes, B may still fail if the feature or functionality it tests is erroneous.

A test case which is not dependent on any other test cases is referred to as an *independent test case* which will not fail due to the failure of another test case. According to our definition for the dependency relation we classify test cases in the following groups:

- **First Class Test Cases (white nodes):** Independent test cases.
- **Second Class Test Cases (gray nodes):** Those which are dependent on one or more independent test cases.
- **Third Class Test Cases (black nodes):** Those which are dependent on at least one dependent test case.

In a multiple dependency relationship two distinct scenarios can exist:

- Passing of both test case A and B is necessary (but not enough) for C to succeed. In other words, if any of A or B fails, it can be concluded that C will also fail. In this case, based on the result of A and B, test case C will not be chosen as a candidate for execution. We refer to this as an AND dependency relationship, which can be formulated by Boolean operators:
  if $result(A) = pass \wedge result(B) = pass \rightarrow$ consider C for execution.
- Passing of A or B is enough so that C is selected as a candidate for execution (implying that C also has a chance to pass). In fact, only if both A and B fail, then it can be concluded that C will also fail, and therefore, will not be chosen for execution. This is regarded as an OR dependency relationship in this paper:
  if $result(A) = pass \vee result(B) = pass \rightarrow C$ can be executed (alternatively: if $result(A) = fail \wedge result(B) = fail \rightarrow$ do NOT consider C for execution).

III. Approach

Our approach for test case prioritization and selection is based on the valuation of both *dependency degree* and also other test case attributes. The main objective of the approach is to evaluate the effect of test case dependencies in selection and ordering of test cases such that redundant test cases are avoided during execution. To this end, we assume that there is information on test case dependencies corresponding to a binary relation between test cases. Figure 2 shows an overall view of the approach.

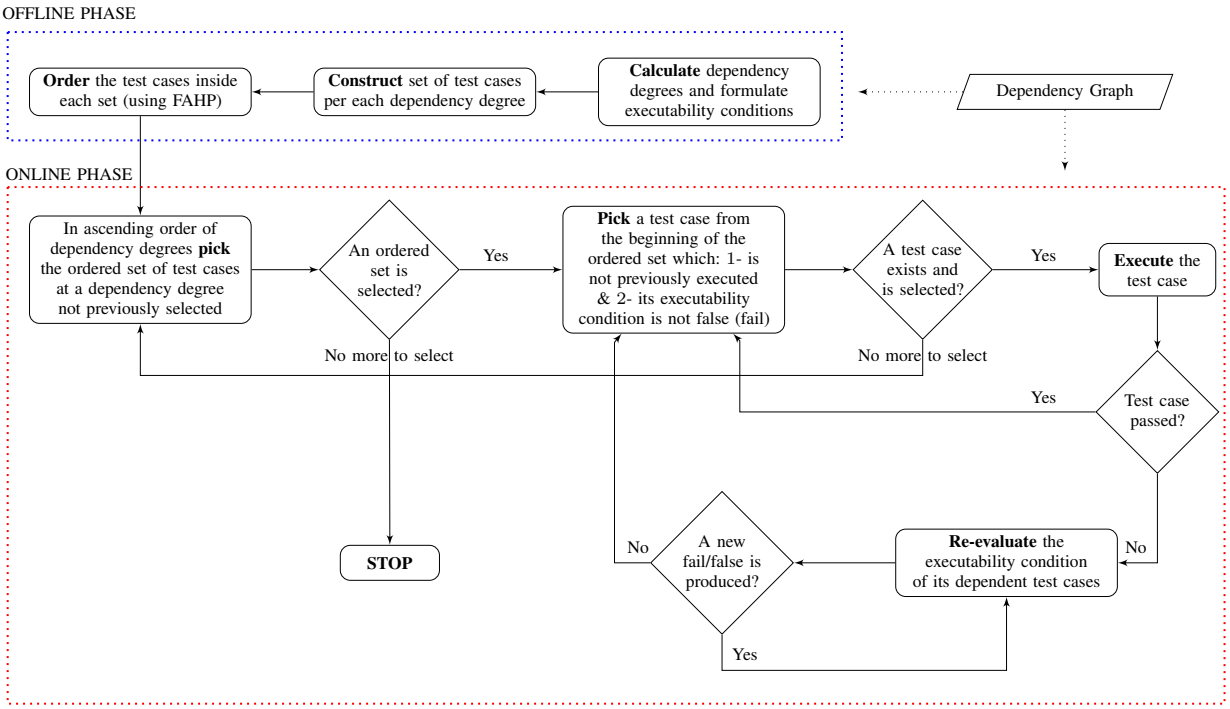


Fig. 2: The steps of the proposed approach

The proposed approach consists of two phases: offline and online. In the offline phase, a *dependency degree* for each test case in relation to all other test cases is calculated. As the result, some test cases might have the same *dependency degrees*. In this step, test cases with the same *dependency degrees*, would be prioritized by applying FAHP. Considering an ascending order for *dependency degrees* and their corresponding set of test cases, an offline order for selection and execution of test cases can be determined. These sets (prioritized in an ascending order based on their *dependency degrees*) are then used in the online phase of the approach. Now the prioritized test cases are ready for execution. The results (pass or fail) for every single execution would be monitored in the online phase. In this phase it is decided that based on the verdict of a test case (pass or fail), which test case should be chosen for the next execution. By establishing and consulting the dependency relations between test cases, we are able to run them in an order that results in avoiding redundancy, and thus, a more efficient use of test execution resources.

A. Dependency Degree

Based on the dependency relationships between test cases, a dependency graph is constructed that represents test cases as nodes and the dependency relationships as directed edges. For each node in this graph, a *dependency degree* value is calculated as follows:

1) The *dependency degree* of independent nodes (with no incoming edges) is set as 1.
2) For each directed edge (e) outgoing from a node, a value as its weight (W_e) (hence a weighted directed graph) is assigned which is calculated as:

$$W_e = D_{source\_node} + 1 \tag{1}$$

where D_{source_node} represents the *dependency degree* of the node at the start of the edge.
3) The weight of the output edge (W_o) of an AND gate will be the the maximum of the weights of the incoming edges to the gate:

$$W_o = \text{Max}\{W_i\} \tag{2}$$

4) The weight of the output edge (W_o) of an OR gate will be the the minimum of the weights of the incoming edges to the gate:

$$W_o = \min\{W_i\} \tag{3}$$

5) The *dependency degree* of a node (v) will be the weight of the incoming edge (e) to it (either directly from another node or from an AND or OR gate):

$$D_v = W_e \tag{4}$$

Considering that the dependencies of test cases can be complex as described above, we also introduce the concept of *executability condition* for each test case and node in the graph. *Executability condition* of a node is the logical condition that is resulted from the incoming edges to that node. We use the *executability condition* to reflect when a test case needs not

be considered for execution based on the fail result of other test cases it is dependent on. In this context, the pass result of a test case will be equivalent to the logical true, and the fail result will be the logical false, and all nodes are assumed to be true by default. Therefore, in Figure 3, the *executability condition* of node D will be $A \vee (B \wedge C)$.

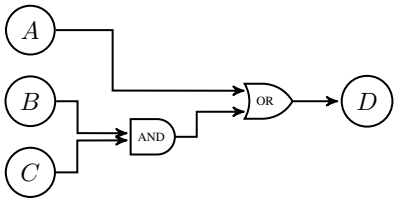


Fig. 3: An illustration of executability condition

In this case, we can determine to skip executing test case D, only when test case A and either of test case B or C have failed (considering the OR relation between test case A and the AND relation that groups test case B and C). However, if for example, only test case A has failed, the *executability condition* of D can still become true, implying that there is still one more way (through the AND relation) that has to be fail (i.e., false) until we can definitely determine that D will also fail. When the *executability condition* of a test case is evaluated as false, that test case can then be skipped and not selected for execution. Evaluation of *executability condition* is done only in the online phase of our approach, while the *executability condition* itself can be determined and formulated in the offline phase.

B. Test Case Prioritization: FAHP

After calculating *dependency degrees*, some test cases can end up having the same *dependency degrees*. In this situation, we prioritize them based on some other criteria (such as requirement coverage, time efficiency, cost efficiency and fault detection probability). In fact, there is no test execution preference for test cases with the same *dependency degree*. The main goal of applying FAHP for prioritizing test cases is, giving more chance for earlier execution to the test cases which satisfy the identified criteria properly. FAHP is not, however, limited to any particular set of criteria and in different systems and contexts users can have their own set of criteria. For computing the effects of the criteria on the test cases, we define a set of linguistics variables (e.g., low, high, etc.) and then questionnaires are sent to testers, where testers specify the values for each criterion. The answers of the questionnaire are then interpreted into fuzzy environment. By re-defining AHP in fuzzy environment (called FAHP), the approach is more practical in real world scenarios when precise quantified values cannot be given for each criterion [9].

Fuzzy truth represents membership in vaguely defined sets. Variables over these sets are called fuzzy variables. From a user perspective, fuzzy properties are often described using linguistic variables. This section outlines the process of transforming a linguistic value into a fuzzy value. In this paper we use five triangular-shaped membership functions, shown in Figure 4.

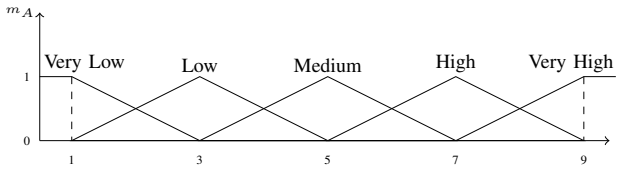


Fig. 4: Fuzzy membership functions for the linguistic variables

Definition 2: A triangular fuzzy number *(TFN)* can be defined as a triplet $M = (l, m, u)$ where l, m, u are real numbers and l indicates low bound, m is modal and u represents a high bound (see [10]).

By using Table I, we are able to interpret the linguistic variables in the form of *TFN*s.

TABLE I: THE FUZZY SCALE OF IMPORTANCE

Fuzzy number	Description	Triangular fuzzy scale	Domain	$m_A(x)$
$\tilde{9}$	Very High	$(7, 9, 9)$	$7 \leq x \leq 9$	$(x-7)/(9-7)$
$\tilde{7}$	High	$(5, 7, 9)$	$7 \leq x \leq 9$	$(9-x)/(9-7)$
			$5 \leq x \leq 7$	$(x-5)/(7-5)$
$\tilde{5}$	Medium	$(3, 5, 7)$	$5 \leq x \leq 7$	$(7-x)/(7-5)$
			$3 \leq x \leq 5$	$(x-3)/(5-3)$
$\tilde{3}$	Low	$(1, 3, 5)$	$3 \leq x \leq 5$	$(5-x)/(5-3)$
			$1 \leq x \leq 3$	$(x-1)/(3-1)$
$\tilde{1}$	Very Low	$(1, 1, 3)$	$1 \leq x \leq 3$	$(3-x)/(3-1)$

The fuzzy comparison matrix $A = (\tilde{a}_{ij})_{n \times n}$ can be formulated and structured as [11]:

$$A = \begin{pmatrix} (111) & \tilde{a}_{12} & \dots & \tilde{a}_{1n} \\ \tilde{a}_{21} & (111) & \dots & \tilde{a}_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \tilde{a}_{n1} & \tilde{a}_{n2} & \dots & (111) \end{pmatrix} \quad (5)$$

where $\tilde{a}_{ij}$ $(i = 1, 2, ..., n, j = 1, 2, ..., m)$ is an element of the comparison matrix and the reciprocal property of the comparison matrix is defined as $\tilde{a}_{ij} = \tilde{a}_{ij}^{-1}$. The pairwise comparisons need to be applied on every criteria and alternative, and the values for $\tilde{a}_{ij}$ come from a predefined set of fuzzy scale value as showed in Table I. Moreover $\tilde{a}_{ij}$ represents a *TFN* in the form of $\tilde{a}_{ij} = (l_{ij}, m_{ij}, u_{ij})$ and matrix A consists of the following fuzzy numbers:

$$\tilde{a}_{ij} = \begin{cases} 1 & i = j \\ \tilde{1}, \tilde{3}, \tilde{5}, \tilde{7}, \tilde{9} \quad or \quad \tilde{1}^{-1}, \tilde{3}^{-1}, \tilde{5}^{-1}, \tilde{7}^{-1}, \tilde{9}^{-1} & i \neq j \end{cases}$$

For computing a priority vector of matrix A, we need to calculate the value of fuzzy synthetic extent $\tilde{S}_i$ for each row in matrix A by (see [10]):

$$\tilde{S}_i = \sum_{j=1}^{m} \tilde{a}_{ij} \otimes \left[\sum_{i=1}^{n} \sum_{j=1}^{m} \tilde{a}_{ij} \right]^{-1} \quad (6)$$

where $\tilde{a}_{ij}$ is a *TFN*, $\otimes$ is the fuzzy multiplication operator. The degree of possibility for a convex fuzzy number can then be calculated by:

$$V(\tilde{a}_2 \geq \tilde{a}_1) = \text{hgt}(\tilde{a}_1 \cap \tilde{a}_2) = \frac{l_1 - u_2}{m_2 - u_2 + m_1 - l_1} = d \quad (7)$$

where d is the ordinate of the highest intersection point between $\tilde{a}_1$ and $\tilde{a}_2$ and the term hgt indicates the height of fuzzy numbers on the intersection of $\tilde{a}_1$ and $\tilde{a}_2$ (see [10]). As last step, we measure the weight vector for the criteria, assuming:

$$d'(A_i) = \min V(\tilde{S}_i \geq \tilde{S}_k), k = 1, 2, ..., n, k \neq i$$

where $A_i (i = 1, 2, \ldots, m)$ is the m decision alternative and n is the number of criteria, then the weight vector is obtained by (see [10]):

$$W'(A_i) = (d'(A_1), d'(A_2), ..., d'(A_m))^T, A_i (i = 1, 2, ..., m) \quad (8)$$

the normalized weight vectors can be calculated via normalizing Eq. (8) (see [12]):

$$W(A_i) = (d(A_1), d(A_2), ..., d(A_n))^T \quad (9)$$

where W is a non-fuzzy number and represents the arrangement of the alternatives. The importance degree of a criterion (W_{C_j}) can be calculated by:

$$W_{C_j} = \frac{W(A_j)}{\sum_{i=1}^{n} W(A_i)}, j = 1, ..., n \quad (10)$$

C. Offline and online phases

In this section, through an example, we show how the calculation of *dependency degree* is done where we have both AND, OR situations and 13 test cases that test the system under test. Figure 5 illustrates a sample calculated dependency graph for the test cases where the calculated *dependency degree* for each node is specified inside parenthesis and weight of each edge is shown above it.

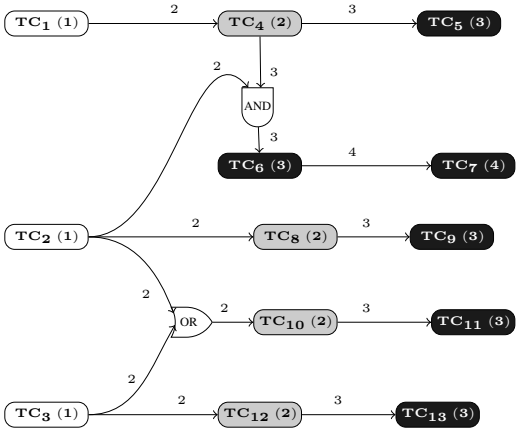


Fig. 5: Dependency Graph

By using Eqs. 1, 2, 3, and 4 we get the following *dependency degrees* for the test cases. Noting that the *dependency degrees* for each independent node is equal to 1 then $D_{TC_1} = 1$. For calculating the *dependency degrees* for the next node in the first row, which is the grey node TC_4, first the weight of the incoming edge to this node is calculated using Eq. 1: $W_{TC_4} = D_{source\_node} + 1 = D_{TC_1} + 1 = 2$.

Since there is no other incoming edge to TC_4, Eq. 4 is applied and therefore $D_{TC_4} = 2$ (the weight and value coming from the edge between TC_1 and TC_4). Similarly, for node D_{TC_6} first the weights of the incoming edges are calculated using Eq. 1. Then because of the AND relation between the incoming edges, Eq. 2 is applied for calculating the *dependency degree* for node D_{TC_6}:

$$D_{TC_6} = Max\{2, 3\} = 3$$

The set of test cases with the same *dependency degree* can be further prioritized by FAHP according to a selection of criteria (cost, execution time, etc). The result of this step will be an ordered set of test cases with the same *dependency degree*. A sample output as illustrated in Table II is produced.

TABLE II: ORDERED SET OF TEST CASES PER DEPENDENCY DEGREE BY FAHP

Dependency Degree	Set of ordered test cases
1	$\{TC_2, TC_3, TC_1\}$
2	$\{TC_8, TC_{10}, TC_{12}, TC_4\}$
3	$\{TC_9, TC_6, TC_{11}, TC_5, TC_{13}\}$
4	$\{TC_7\}$

Having an ascending oder of test cases for each *dependency degree*, an offline order for execution of test cases is generated. This means that starting from the lowest calculated *dependency degree*, the test cases can be selected for execution in the order that is determined for them using FAHP (i.e., ordered set). This can be repeated for the subsequent *dependency degrees*. In the online phase, the result of each executed test case is also added to the ordering process.

In the online phase, the result of each test case execution is also taken into account in the selection of the next test case(s) for execution. The steps that are performed in the online phase are as follows: the first item in the set of test cases from the lowest *dependency degree* is selected and executed. Then the next item in the same set is executed until there is no item left. Then the (ordered set of test cases in the) next *dependency degree* greater than the previously selected *dependency degree* is considered. During the whole process, after the execution of each test case and based on its result, the *executability condition* of the test cases that are dependent on it are (re-) evaluated. In selecting test cases from the ordered set of test cases at each *dependency degree*, if the *executability condition* of a test case is false, it will be skipped and not selected for execution.

IV. INDUSTRIAL CASE STUDY

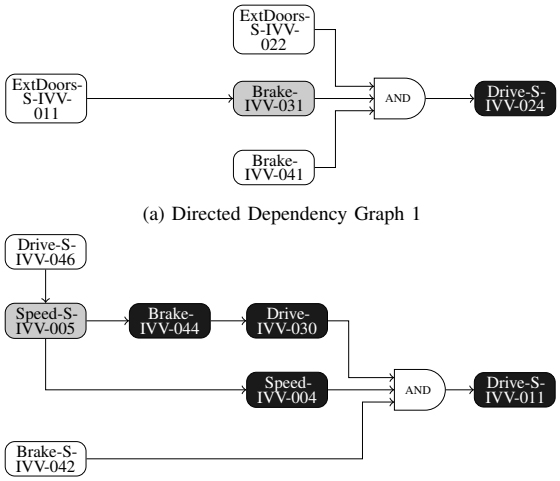
We have started validating our approach at Bombardier Transportation AB (BT) in Sweden. BT develops and manufactures trains and railway equipment. Reliability and safety of the train control management system along with all integrated functions is of great importance for BT. We plan to conduct a series of case studies to continuously adapt and improve our approach for BT. Case study represents a good choice as a research method because we need to develop a deeper understanding of decisions impacting test efficiency at BT. Furthermore, as our

final objective is to improve the current state of testing practice at BT and it may involve different kinds of evidence, case study research is further justified [13]. This section presents the results of a case study where we evaluate the feasibility of our proposed approach. The objective of this case study is to understand the existing order of test execution at BT and how our approach is expected to impact test efficiency. We have selected a running project at BT as our case. The project is selected to fit the case study objectives as we wanted to observe and track the order of test execution. Moreover, our units of analysis is limited to two sub-level function groups (SLFGs): *brake system* and *air supply*. A SLFG is a grouping of functions related to a key functional requirement; other examples of SLFGs include aerodynamic performance, propulsion and auxiliary power. *Brake system* and *air supply* SLFGs were selected as a matter of convenience since the test cases for them were ready to be executed as part of the running project at BT. Moreover these SLFGs represent two of the critical function groups in a train control management system, having inter-dependencies and these must be tested. Our current context is limited to a set of 12 integration test cases only, but these test cases are expensive to run in terms of time (approximately 1 hour per test case) since they cover coarse requirements. Moreover these test cases are run at a sub-system level, meaning that they are more time consuming to run than tests at unit level [14]. On a limited number of expensive simulators, therefore re-running them due to unintended failures is costly as simulators are kept busy waiting for other test cases to execute. Table III list down the test cases used in this case study along with their associated SLFG. We have retained the test case IDs used in BT for brevity.

TABLE III: TEST CASE IDS WITH ASSOCIATED SLFG

No.	Test case ID	Associated SLFG
1	Drive-S-IVV-046	Brake system
2	Speed-S-IVV-005	Brake system
3	ExtDoors-S-IVV-011	Air supply
4	ExtDoors-S-IVV-022	Air supply
5	Brake-IVV-031	Brake system
6	Brake-IVV-041	Brake system
7	Drive-S-IVV-024	Air supply
8	Speed-IVV-004	Air supply
9	Drive-IVV-030	Brake system
10	Brake-IVV-044	Brake system
11	Brake-S-IVV-042	Brake system
12	Drive-S-IVV-011	Air supply

The data collection for the case study was done using participant observation, questionnaire as well as taking help from archival data for finding the cause of test case failures. As is shown in Figure 2, our approach is usable in two phases: offline and online. In the beginning of the offline phase, a test expert at BT answered a questionnaire where the test dependencies were identified based on requirements. The mapping of these dependencies resulted in two dependency graphs as shown in Figure 6. As given in Section II-A, the white, grey and black nodes in Figure 6 show first class (independent), second class and third class test cases. Also in the current set of test cases, we only have AND situations but no OR situation.



(a) Directed Dependency Graph 1

(b) Directed Dependency Graph 2

Fig. 6: Directed dependency graphs for Brake system and Air supply SLFGs

Using Eqs. 1, 2 & 4, the *dependency degree* for each test case is also calculated, given in Table IV.

TABLE IV: SET OF TEST CASES PER DEPENDENCY DEGREE

Dependency Degree	Set of test cases
1	{Drive-S-IVV-046, Brake-S-IVV-042, ExtDoors-S-IVV-022, Brake-IVV-041, ExtDoors-S-IVV-011}
2	{Speed-S-IVV-005, Brake-IVV-031}
3	{Brake-IVV-044, Speed-IVV-004, Drive-S-IVV-024}
4	{Drive-IVV-030}
5	{Drive-S-IVV-011}

As we can see in Table IV, there is more than one test case with *dependency degrees* 1, 2 and 3. To select the best candidates for execution in the online phase, we need to prioritize test cases having same *dependency degree*, based on an existing criteria. As explained in Section III-B, we propose FAHP for prioritizing test cases in this step. In discussions with the test expert at BT, the following criteria have been identified, sorted in descending order of preference for BT:

- **Requirements coverage:** Refers to the number of requirements tested by a test case.
- **Time efficiency:** Is the sum of test case creation time, test case execution time and test environment setup time.
- **Cost efficiency:** Refers to the cost incurred by BT in test case configuration (e.g., setting environment parameters, hardware setup) and test case implementation.
- **Fault detection probability:** Refers to the average probability of detecting a fault by each test case.

We need to reiterate that these criteria have different preferences for BT, with requirements coverage being the most important criterion at sub-system level testing. The resulting weights for the mentioned criteria, as calculated through pairwise comparisons between the criteria, are shown in Table V.

TABLE V: PAIRWISE COMPARISONS OF CRITERIA

Rank	Criteria	Priority
1	Requirement Coverage	67.5 %
2	Time Efficiency	22.5 %
3	Cost Efficiency	7.5 %
4	Fault Detection Probability	2.5 %

While there is a possibility to achieve quantitative numbers on some criteria, e.g., requirements coverage, there is always an element of human judgment in estimating them. In order to get expert judgment on these criteria for our set of test cases, five linguistic variables (Figure 4) are defined. A questionnaire was designed where the test experts responded with, for each test case, a linguistic variable for the different criteria. Table VI represents a sample survey questionnaire which has been sent to the test experts at BT, the variables were assigned using pair-wise comparisons between the criteria.

TABLE VI: A SAMPLE WITH VALUES VERY LOW(VL), LOW (L), MEDIUM (M), HIGH (H) AND VERY HIGH (VH)

Test Case ID	Requirement Coverage	Time	Cost	Fault Detection
Drive-S-IVV-046	VH	H	L	M
Speed-S-IVV-005	M	VL	M	M
ExtDoors-S-IVV-011	VL	H	H	L
ExtDoors-S-IVV-022	H	L	M	L
Brake-IVV-031	VL	M	M	L
Brake-IVV-041	VL	L	M	M
Drive-S-IVV-024	L	H	H	M
Speed-IVV-004	L	M	M	M
Drive-IVV-030	L	H	L	L
Brake-S-IVV-044	L	H	L	H
Brake-S-IVV-042	VL	H	M	M
Drive-S-IVV-011	VL	M	M	H

The linguistic variables have been interpreted in a set of fuzzy numbers. The last step in the offline phase of our approach involves prioritizing test cases with the same *dependency degree* by using Eqs. (6) to (10). The results are shown in Table VII:

TABLE VII: ORDERED SET OF TEST CASES BY FAHP

Dependency Degree	Ordered set of test cases (FAHP)
1	{Drive-S-IVV-046, ExtDoors-S-IVV-011, ExtDoors-S-IVV-022, Brake-S-IVV-042, Brake-IVV-041}
2	{Speed-S-IVV-005, Brake-IVV-031}
3	{Drive-S-IVV-024, Brake-IVV-044, Speed-IVV-004}
4	{Drive-IVV-030}
5	{Drive-S-IVV-011}

We now have an order of execution of the test cases that takes into account test case dependencies along with multiple criteria of importance for BT.

A. Preliminary results of online evaluation

The objective with online evaluation is to identify improvement potential in the current ordering of test executions at BT and to assess if the online phase of our approach will be of any benefit. So far, we have monitored and observed the execution of a subset of our 12 test cases and the results have given us an early indication of usefulness of our approach.

The subset of tests monitored are: *Drive-S-IVV-024*, *ExtDoors-S-IVV-022*, *Brake-IVV-031*, *Brake-IVV-041* and *ExtDoors-S-IVV-011*, shown in Figure 6a. It should be noted that the current way of executing these tests at BT does not follow a dependency structure, rather the tester selects a test case to execute based on intuition and knowledge regarding if the associated functionality has been implemented as yet. The tester has to configure the simulator in an effort to successfully run a test case which also includes configuration of any signal inputs that are expected as part of dependencies between test cases. As will be evident shortly, without any systematic way to identify these dependent signals, the current execution of test cases need multiple runs which is both time consuming and expensive for BT.

We continued monitoring test execution until every test case had a pass verdict. Table VIII presents the results of four runs of test execution that were required to successfully execute the test cases, while also showing the order of execution.

TABLE VIII: EXECUTION (EXEC.) ORDER - BT

Exec. Order	Test Case ID	Exec.1	Exec.2	Exec.3	Exec.4
1	Drive-S-IVV-024	Fail	Fail	Fail	Pass
2	Brake-IVV-031	Fail	Fail	Pass	—
3	ExtDoors-S-IVV-011	Fail	Pass	—	—
4	ExtDoors-S-IVV-022	Not Run	Fail	Pass	—
5	Brake-IVV-041	Fail	Pass	—	—

It is evident that the execution order (column 1 in Table VIII) has not followed the dependency directed graph as shown in Figure 6a. In the first execution run (column 3, Table VIII), the first test case executed is *Drive-S-IVV-024*. According to our calculation, the *dependency degree* for this test case is 3 (see Table IV), which means that it is a dependent test case and its successful execution is dependent on the successful execution of prior test case(s). This test case failed in the first execution run as shown as 'Fail' in Table VIII (column 3). The reason for this failure could be that it found a fault but reading the logged test record reveals that it failed because 'the door lock status failed', which would have been tested earlier by test case *ExtDoors-S-IVV-022*. While this was the reason mentioned in the test records, we know from Figure 6a that the successful executions of *Drive-S-IVV-024* also depends on successful execution of two other test cases (*Brake-IVV-031 & Brake-IVV-041*). This is the reason why the test case *Drive-S-IVV-024* does not pass until the fourth test run, after the test cases it depends on have successfully passed. If the tester had known the correct dependency structure among test cases, wasted effort in running *Drive-S-IVV-024* thrice would have been saved. We also measured the test execution time for a single test case; it took approximately one hour to get a verdict (pass or fail). Considering this time, just for testing the test case *Drive-S-IVV-024*, three hours were wasted. The test record for *Brake-IVV-031* in first execution showed that this test case failed because of the 'signal service brake failure'. The test case specification for *ExtDoors-S-IVV-011* explains that this test case tests the signal service brake as well. This is also evident in Figure 6a where *Brake-IVV-031* is dependent on successful execution of *ExtDoors-S-IVV-011*. Thus it turned out to be a wasted effort in executing *Brake-IVV-031* before having a pass verdict on *ExtDoors-S-IVV-011*. In other words, it is a redundant test case to execute. According to our calculations

in Table VII, the *dependency degree* for *ExtDoors-S-IVV-011*, *ExtDoors-S-IVV-022* and *Brake-IVV-041* is 1, indicating that these are independent test cases.In the first execution run, none of these independent test cases were able to get a pass verdict, due to reasons attributed to faults in test specifications. For *ExtDoors-S-IVV-022*, the test case could not even get started (indicated as 'Not Run' in Table VIII) while for *ExtDoors-S-IVV-011* and *Brake-IVV-031*, failures resulted after the test cases had run for approximately one hour each. For *ExtDoors-S-IVV-022*, when the fault in the test specification was fixed to enable it to run in the second execution run, it failed again due to another fault in the test specification. This highlights improvement opportunities in design of test specifications at BT but it is not a focus in this paper.

In second execution run, the other two independent test cases (*ExtDoors-S-IVV-011* and *Brake-IVV-041*) were able to get a pass verdict which allowed *Brake-IVV-031* to pass in the third execution run. *ExtDoors-S-IVV-022* was also eventually passed in third execution once the problem in the test specifications was fixed. The already passed test cases are now represented with '—' in Table VIII. The test case *Drive-S-IVV-024* was also eventually passed in the fourth execution run once the test cases it was dependent on were passed. These are only preliminary results but they have given us evidence that much time can be saved by incorporating dependency information in ordering test execution. The total estimated time for executing test cases in Figure 6 is approximately 5 hours (one hour per test case). But the total time taken to execute test cases in Table VIII is 45 hours. We need to consider that re-running a test case has additional associated costs, such as troubleshooting the cause of failure, potential update of test case, implantation, restarting the simulator and potential configuration setting. In this case, 40 hours of testing time was wasted. Given it takes approximately 0.5 hour to find dependencies in our case, 39.5 hours of testing time could potentially be saved from using our method. Our proposed approach further recommends an ordering of test cases that have the same *dependency degree*. This promises to further cut down test costs. The early results presented here suggest that efficiency gains can be made using our approach. We, however, need to provide further quantitative evidence in support of our approach by executing the online phase, which is left as a future work.

V. Discussion & Future Extensions

In our proposed approach, test cases were first categorized based on their *dependency degree* resulting in sets of test cases for each *dependency degree*. As each set can contain one or more test cases, FAHP was introduced to prioritize test cases at each *dependency degree*. This prioritization was based on a set of test case attributes serving as criteria in the decision making process. From this perspective, dependency was used in our work as a separate criterion. An alternative way is to use dependencies directly as another criterion in the decision making process. One can also consider using fuzzy dependency relationships. In other words, in our current approach, a test case is either independent or not (i.e., binary: 0 or 1). By including concepts from fuzzy logic, the strength of the dependency between test cases can be specified with fuzzy variables mapping to values over the interval $[0, 1]$. This idea can particularly be helpful in cases where test engineers cannot determine if two test cases are fully dependent or not. To visualize the dependencies of test cases, a directed graph is used in this paper. However, we did not modify the structure of the graph after it was constructed. Another possible extension could be to update the graph dynamically during test execution (e.g., by removing some edges). Regarding the use of graph in providing visual hints to testers, we grouped test cases into three classes with respect to their dependency relations (white, grey, and black nodes). We believe this is a useful basis for discussions in a testing team, not only for dependency issues but potentially also for resolving traceability issues. One interesting future direction is to investigate the opposite form of result dependency. In other words, while here we determined redundancy of test cases based on the fail result of other test cases (i.e., fail based on fail), it would be interesting to consider whether and how from the pass result of a test case, it can similarly be asserted that the result of some other test cases will have to be pass as well (i.e., pass based on pass). So, in our current work, we start by test cases with lowest *dependency degree* and move to the ones with higher *dependency degree* while considering which fail verdicts will result in the failure of other dependent test cases. For the opposite case, test cases might be considered from the highest *dependency degree* towards the ones with lower *dependency degree* and determining their verdict whenever a test case further in the dependency path has passed. A combination of these two approaches (i.e., fail based on fail & pass based on pass) will be another possible future direction of this research.

The approach is generic and independent of the type of analysis performed to identify dependencies. Currently, we are identifying individual dependencies by interviewing text experts. However, this approach may not be feasible when the number of test cases are larger. As a next step, we are therefore considering analysing dependencies based on a combination of temporal order and pattern matching applied to historic test record data.

A. Delimitations

In discussions with BT, four prioritization criteria were agreed upon. But there can be other applicable criteria, e.g., requirements volatility. The increase in the number of criteria is not a limitation of our proposed FAHP approach but it might take more time for pair-wise comparisons. We did not undertake such an analysis in this study. Also the answers to the criteria were given by one test expert at BT. There is a risk that another test expert will give different ratings on the criteria, leading to a different prioritization of test cases. However, we minimized this risk by having an experienced test expert who has a long background in testing train control management systems. We have used triangular fuzzy membership function for evaluating the effect of the identified criteria on each test case. We did not compare other membership functions, e.g., bell-shaped

that might produce a better prioritization of test cases. We used result dependency (fail based on fail) for creating the dependency model. If in a different context, another type of dependency such as state dependency is considered and is more relevant, the approach might not be applicable as it is, and might require some modifications. Moreover, our approach assumes that test case dependencies are identified, either manually or otherwise. We did not assess the cost of identifying these dependencies but in cases where more complex dependencies exist, an automatic inference and extraction of dependencies is more feasible, see e.g., Arlt and Morciniec [8].

VI. Related Work

Use of dependency information to prioritize, select and minimize test suites has recently received much attention. Bates and Horwitz [15] use program dependence graphs and slicing with test adequacy data to identify components of the modified program that can be tested using files from the old test suite and to avoid unproductive retesting of unaffected components. Rothermel and Harrold [16] also used slicing with program dependence graph to identify changed def.-use pairs for regression testing. Our approach uses a black-box approach in the sense that it is independent of the source-code modifications. We do not have access to implementation details of functions which is realistic for testing at higher levels. Also we do not address regression testing in particular.

Ryser and Glinz [17] propose the use of dependency charts to manage dependencies between scenarios for systematically developing test cases for system test. They differentiate between three types of dependencies: abstraction, temporal and causal, while data and resource dependencies are taken as special cases of causal dependencies. Test cases are derived by traversing paths in the dependency chart, taking into account data and resource annotations and other specified conditions. While test suite reduction or prioritization is not their objective, their approach shows the importance of managing dependencies and interrelations between scenarios for thorough system testing, e.g., trying to break constraints and restrictions.

Zhang et al. [18] challenge the test independence assumption of much of the traditional regression test prioritization (e.g. [19], [20], [21]) and test selection (e.g. [22], [23], [24], [25]) approaches. This assumption stems from the *controlled regression testing assumption* [26] which states: given a program P and a modified version P', when P' is tested with test t, all factors that may influence the outcome of this test remain constant, except for the modified code in P'. Zhang et al. [18] show that dependent test cases affect the output of five test case prioritization techniques. They further implemented four algorithms to detect dependent tests. An empirical study of 96 real-world dependent tests from 5 software issue tracking systems showed that dependent tests do arise in practice, both for human-written or automatically-generated tests. The presence of dependencies between tests is also confirmed by Bell et al. [27] and Lou et al. [28]. Haidry and Miller [29] use dependency structure of test cases, in the form of a directed acyclic graph, to prioritize test cases. The test cases are prioritized based on different forms of graph coverage values, however a set of independent tests are arbitrarily prioritized, which leads to lower performance in case of fewer unconnected tests. The authors emphasize the need to combine dependency with other types of information to improve test prioritization. Our work contributes to fill this gap whereby test case dependencies along with a number of other criteria are used to prioritize test cases.

Caliebe et al. [30] present an approach based on dependencies between components whereby analysis could be performed on a graph representation of such dependencies. Two applications of their proposed approach are possible: general test case selection and test case prioritization for regression testing. Arlt et al. [8] use logical dependencies between requirements written in a structured format to automatically detect a redundant test case. Their approach is essentially a test suite reduction technique based on the current status of successful tests and failed tests. While being similar in purpose, our focus is mainly on the steps after the identification of dependencies. Moreover, our proposed approach cover a more general form of dependencies that can address more complex scenarios consisting of AND and OR relations.

There has also been previous work on using fuzzy computing approaches for multi-faceted test case fitness evaluation, prioritization and selection. Kumar et al. [31] use a fuzzy similarity measure to filter out unfit and high ambiguity test cases based on four parameters of statement coverage, branch coverage, fault detection capability and execution time. Tahvili et al. [9] formulate test prioritization as a multi-criteria decision making problem and apply analytic hierarchy process (AHP) in a fuzzy environment to prioritize test cases. Alakeel [32] present a test case prioritization approach that uses fuzzy logic to measure the effectiveness of a given test in violating program assertions of modified programs while Malz et al. [33] combine software agents and fuzzy logic for automated prioritization of system test cases. Xu et al. [34] use a fuzzy expert system to prioritize test cases based on knowledge represented by customer profile, analysis of past test case results, system failure rate, and change in architecture. A similar approach is used by Hettiarachchi et al. [35] where requirements risk indicators such as requirements modification status, complexity, security, and size of the software requirements are used in a fuzzy expert system to prioritize test cases. Schwartz and Do [36] use a fuzzy expert system to choose the most cost-effective regression testing technique for regression testing sessions. While similar to these studies in the use of a fuzzy approach, this paper is unique in combining dependencies in test cases with multiple criteria.

VII. Summary & Conclusion

In this paper, we provide the following main contributions: (1) we formally define the *dependency degree* as a metric to be used in test case prioritization, together with an algorithm for calculating it; (2) we introduce a new approach for dynamic test case selection using the result of executed test cases and their *dependency degrees* whereby an offline order based on

dependency of test cases is produced along with prioritization of test cases using FAHP; (3) we apply the method to an industrial case study of a safety-critical train control subsystem, compare it to the baseline test case execution order, and give a brief analysis of the results. The results of the BT case study show that the concept of 'fail based on fail' is applicable and can reduce test execution time. When the testers did not follow and consider test case dependency relations, some test cases were selected which failed due to the failure of test cases they were depending on. Consequently, using our approach will enable higher test execution efficiency by identifying and avoiding such forms of test redundancies.

ACKNOWLEDGEMENTS

This work was supported by VINNOVA grant 2014-03397 (IMPRINT) and the Swedish Knowledge Foundation (KKS) grants 20130085 (TOCSYC) and ITS-EASY industrial research school. Special thanks to Anders Skytt, Ola Sellin and Kjell Bystedt at Bombardier Transportation, Västerås-Sweden.

REFERENCES

[1] M. Saadatmand and M. Sjödin, "On combining model-based analysis and testing," in *Proceedings of the 10th International Conference on Information Technology: New Generations*, 2013.

[2] S. Tahvili, W. Afzal, M. Saadatmand, M. Bohlin, D. Sundmark, and S. Larsson, "Towards earlier fault detection by value-driven prioritization of test cases using fuzzy TOPSIS," in *Proc. of the 13th International Conference on Information Technology: New Generations*, 2016.

[3] A. Srivastava, J. Thiagarajan, and C. Schertz, "Efficient integration testing using dependency analysis," *Tech. Report; no. MSR-TR-2005-94*, 2005.

[4] C. Carlsson and R. Fuller, "Fuzzy multiple criteria decision making: Recent developments," *Fuzzy Sets and Systems*, pp. 415–437, 1996.

[5] A. K. Onoma, W.-T. Tsai, M. Poonawala, and H. Suganuma, "Regression testing in an industrial environment," *Communications of the ACM*, vol. 41, no. 5, pp. 81–86, 1998.

[6] H. Agrawal, J. R. Horgan, E. W. Krauser, and S. London, "Incremental regression testing." in *ICSM*, vol. 93. Citeseer, 1993, pp. 348–357.

[7] International Software Testing Qualifications Board, *Standard glossary of terms used in software testing*, v2.4 ed., ISTQB, 2014.

[8] S. Arlt, T. Morciniec, A. Podelski, and S. Wagner, "If A fails, can B still succeed? Inferring dependencies between test results in automotive system testing," in *Proceedings of the 8th IEEE International Conference on Software Testing, Verification and Validation*, 2015.

[9] S. Tahvili, M. Saadatmand, and M. Bohlin, "Multi-criteria test case prioritization using fuzzy analytic hierarchy process," in *The 10th International Conf. on Software Engineering Advances*, 2015.

[10] Y.-C. Tang and T. W. Lin, "Application of the fuzzy analytic hierarchy process to the lead-free equipment selection decision," *International Journal of Business and Systems Research*, vol. 5, pp. 35–56, 2011.

[11] T. Terano, *Fuzzy engineering toward human friendly systems*, no. v. 2.

[12] D.-Y. Chang, "Applications of the extent analysis method on fuzzy AHP," *European Journal of Operational Research*, vol. 95, pp. 649 – 655, 1996.

[13] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, 2009.

[14] H. Hemmati, L. Briand, A. Arcuri, and S. Ali, "An enhanced test case selection approach for model-based testing: An industrial case study," in *Proceedings of the 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ACM, 2010.

[15] S. Bates and S. Horwitz, "Incremental program testing using program dependence graphs," in *Proceedings of the 20th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 1993.

[16] G. Rothermel and M. J. Harrold, "Selecting tests and identifying test coverage requirements for modified software," in *Proceedings of the International Symposium on Software Testing and Analysis*, 1994.

[17] J. Ryser and M. Glinz, "Using dependency charts to improve scenario-based testing - Management of inter-scenario relationships: Depicting and managing dependencies between scenarios," in *the 17th International Conference on Testing Computer Software*, 2000.

[18] S. Zhang, D. Jalali, J. Wuttke, K. Muşlu, W. Lam, M. D. Ernst, and D. Notkin, "Empirically revisiting the test independence assumption," in *Proceedings of the 2014 International Symposium on Software Testing and Analysis*, 2014.

[19] S. Elbaum, A. G. Malishevsky, and G. Rothermel, "Prioritizing test cases for regression testing," *SIGSOFT Software Engineering Notes*, vol. 25, no. 5, pp. 102–112, 2000.

[20] B. Jiang, Z. Zhang, W. K. Chan, and T. H. Tse, "Adaptive random test case prioritization," in *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2009.

[21] A. Srivastava and J. Thiagarajan, "Effectively prioritizing tests in development environment," *SIGSOFT Software Engineering Notes*, vol. 27, no. 4, pp. 97–106, 2002.

[22] M. J. Harrold, J. A. Jones, T. Li, D. Liang, A. Orso, M. Pennings, S. Sinha, S. A. Spoon, and A. Gujarathi, "Regression test selection for Java software," *SIGPLAN Notes*, vol. 36, no. 11, pp. 312–326, 2001.

[23] A. Nanda, S. Mani, S. Sinha, M. J. Harrold, and A. Orso, "Regression testing in the presence of non-code changes," in *4th IEEE International Conference on Software Testing, Verification and Validation*, 2011.

[24] H.-Y. Hsu and A. Orso, "Mints: A general framework and tool for supporting test-suite minimization," in *Proceedings of the 31st International Conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2009.

[25] A. Orso, N. Shi, and M. J. Harrold, "Scaling regression testing to large software systems," *SIGSOFT Software Engineering Notes*, vol. 29, no. 6, pp. 241–251, 2004.

[26] G. Rothermel and M. J. Harrold, "Analyzing regression test selection techniques," *IEEE Transactions on Software Engineering*, vol. 22, no. 8, pp. 529–551, 1996.

[27] J. Bell, G. Kaiser, E. Melski, and M. Dattatreya, "Efficient dependency detection for safe java test acceleration," in *Proceedings of the 10th Joint Meeting on Foundations of Software Engineering*, 2015.

[28] Q. Luo, F. Hariri, L. Eloussi, and D. Marinov, "An empirical analysis of flaky tests," in *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2014.

[29] S. Haidry and T. Miller, "Using dependency structures for prioritization of functional test suites," *IEEE Transactions on Software Engineering*, vol. 39, no. 2, pp. 258–275, 2013.

[30] P. Caliebe, T. Herpel, and R. German, "Dependency-based test case selection and prioritization in embedded systems," in *Proceedings of the 2012 IEEE 5th International Conference on Software Testing, Verification and Validation*, 2012.

[31] M. Kumar, A. Sharma, and R. Kumar, "Fuzzy entropy-based framework for multi-faceted test case classification and selection: an empirical study," *IET Software*, vol. 8, pp. 103–112(9), 2014.

[32] A. M. Alakeel, "Using fuzzy logic in test case prioritization for regression testing programs with assertions," *The Scientific World J.*, pp. 1–9, 2014.

[33] C. Malz, N. Jazdi, and P. Gohner, "Prioritization of test cases using software agents and fuzzy logic," in *Proceedings of the 5th International Conf. on Software Testing, Verification and Validation*, 2012.

[34] Z. Xu, K. Gao, T. Khoshgoftaar, and N. Seliya, "System regression test planning with a fuzzy expert system," *Inf. Sciences*, vol. 259, 2014.

[35] C. Hettiarachchi, H. Do, and B. Choi, "Risk-based test case prioritization using a fuzzy expert system," *Information and Software Technology*, vol. 69, pp. 1–15, 2016.

[36] A. Schwartz and H. Do, "A fuzzy expert system for cost-effective regression testing strategies," in *Proceedings of the 2013 29th IEEE International Conference on Software Maintenance*, 2013.