A Generic Approach for Modeling Test Case Priorities with Applications for Test Development and Execution

Andreas Hoffmann 1, Axel Rennoch 1,
Ina Schieferdecker 1, Nicole Radziwill 2

1 Fraunhofer FOKUS, Berlin (Germany)
{andreas.hoffmann,axel.rennoch,ina.schieferdecker}@fokus.fraunhofer.de

2 National Radio Astronomy Observatory (NRAO), Charlottesville, Virginia (USA),
nradziwi@nrao.edu

Abstract: This contribution addresses systematic test development methods to include an algorithm to retrieve a test suite execution control in order to run test cases with high priority earlier than others. The approach uses a model that allows both the introduction of user-defined weightings for system features within the test model and an automatic calculation of the test ordering. Based on an algorithm for the calculation of test case weights first results from the application of a tool implementation in pilot projects have been described.

1 Introduction

Software is often designed to be flexible and reconfigurable, so that new and unanticipated business needs can be quickly and easily accommodated. This added dimension of complexity presents additional functional and economic constraints on the quality assurance process. Software adaptation and customization along various parameter sets requires addressing different customization mechanisms such as options, selections, values, or recombination of features. But a comprehensive software testing plan must also adapt to such variability, which requires appropriate weighting and selection of system and test variants. This is the challenge addressed by the new algorithms we present.

Typically, system requirements may be changed along system development due to reduction, extensions, adaptations or other modifications. Hence, it is important to identify those test cases in the test base that are affected by these changes imposing a redefinition of the corresponding test models. The new algorithms support test case priorities based on weights and (feature) potentials (e.g. occurrence, risk, severity indicators). This enables an overall reduction in the magnitude of testing efforts, while minimizing resources needed for test execution.

Our algorithms are applicable in different industrial domains. Tool support has been provided by implementing either new standalone prototype or extending existing well-established software like the Classification Tree Editor, a freely available tool supporting the Classification Tree Method, which is a universal mean for category partition of requirements.

In our paper we summarize the terminology and criteria for test priority techniques understood from white-box testing. Test development techniques from different application domains, including telecommunication and automotive, will be introduced and enhanced in order to consider the determining factors for test priorities (e.g. mathematical probabilities, empirical factors). We conclude by reporting practical results from the application in industrial pilot projects, including better coverage of system requirements and improved early fault detection rates.

2 Related Work

Related work has been found in the white-box testing approaches [4] and [8] that focus on increasing the fault detection rate at earlier test runs (i.e. “faster” rate). The algorithms that have been described differ in:

1. the granularity of the coverage,
2. the basic ordering (e.g. numerical coverage, or consideration of test history),
3. the calculation method approach.

Four granularity types have been identified on different coverage levels: Function, block, decision (branch) and statement level. Since the granularity increases the precision but also the efforts, a cost-optimal compromise has to be found.

In addition to a simple ordering of tests by their numerical coverage of statements or functions, the consideration of a fault exposing potential (FEP), the fault index (FI), and a combination of both (applying FI first and FEP for tests with equal FI) has been proposed. FEP denotes the ratio (approximation) between the number of mutants (to be detected by a test) and the total number of mutants and stands for a high test quality. The FI addresses the history of a test, i.e. how many faults have already been found with that test in previous execution runs.

Finally, the calculation methods may base on a simple calculation of the test ordering that consider all test cases and their potential at once (total calculation) or could be improved due to a stepwise calculation of the next test after removal of the previous, already executed, test cases (additional calculation), i.e. a reset of the ordering after each test selection.

Experimental results have been undertaken and report [4] that any selection technique is better than a random test case execution order. Furthermore, the more advanced methods (FI with FEP, additional calculation method) provide better results than the simpler ones (simple coverage, total calculation method). Due to these results, we tend to introduce the advanced methods in our model, too.

3 Problem description

Finding an optimal test scope for the testing of a complex software system is a very difficult problem. On the one hand, the growth of system complexity causes a dramatic increase of the number of test cases. On the other hand, the time and the budget for testing are limited. To satisfy all of these conditions, the amount of test data has to be reduced to a reasonable list which can be handled within an estimated budget. The priority based test modeling approach helps to solve this problem. It is based on the following points:

1. Analysis of the requirements.
2. Creation the basic system or test model,
3. Derivation the test cases.
4. Reduction/Ordering the test cases by using of weights in the test model.

The goal of the priority based approach is the reduction of the test scope in such way that all functional requirements are covered within the estimated effort budget for testing. The selection of test cases is done in a way which allows finding the most critical errors in the system. Additionally, the test execution is priority based and can be recalculated after each execution step. The more important test cases are executed first and after the execution of each test case it is possible to recalculate the prioritization of the rest of test cases.

4 System and Test Models

The introduction of test case priorities requires a formalized model on the system or test campaign in order to assign concrete values to the system features to be used for the calculation of priorities. In this paper we focus on the existence of concrete values and abstract from the model type, i.e. we assume either a system or test model. In the later case the test model may be subject of a model-based test generation method. In the following we shortly introduce the Classification Tree Method (CTM) as an example for requirement modeling.

CTM is an approach for the systematic design of tests [5]. Only the input and output domains of the system under test are needed for the analysis. The domain of e.g. each input parameter (classification) is systematically segmented into separate subsets, called classes. The test cases are generated by combining classes from different classifications. Thus, every test case constitutes a unique combination of classes.

There is a freely available tool called CTE XL for editing classification trees based on CTM [5], [7] and [11]. With CTE XL, the test case table can be built automatically. The name of the root node can be defined by the user and has only an informal meaning. The tree nodes are classifications, their child nodes are equivalence classes. The test case table can be derived if every test case is due to one equivalence class per classification.

At Nokia Siemens Networks the use of the Classification Tree Method has been used and demonstrated in a modified way. As introduced in previous papers [1] [2] our approach is based on requirement specification defined by using e.g. flow charts that will be mapped to classification trees.

The creation of CTE trees is a straightforward procedure. To do so, the flow chart scenarios need to be mapped to classifications (that represent the behavior junctions) and classes (i.e. alternative branches) in the tree. Sub-trees can be used for sub-requirement specifications. The next figure shows the CTE tree of an example feature.

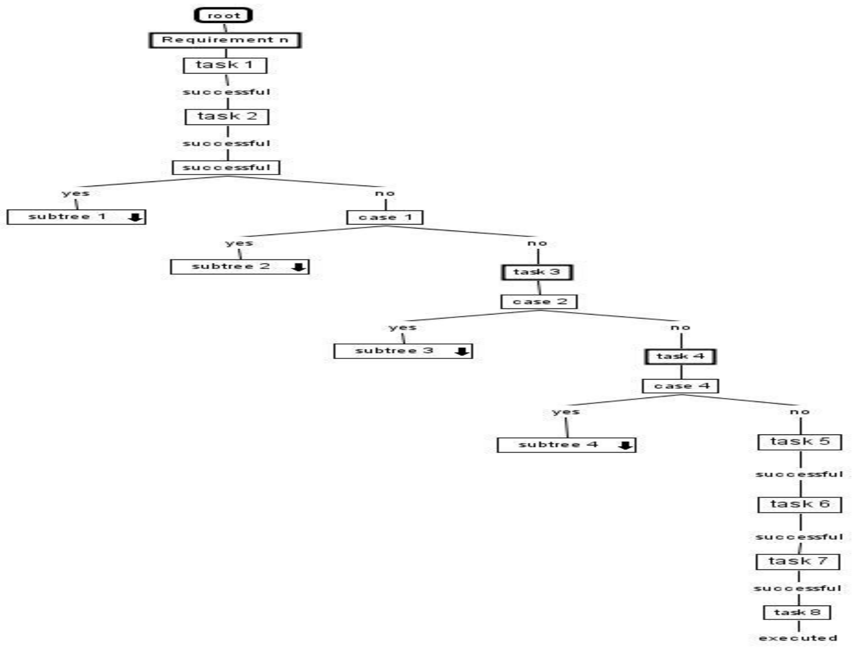


Fig. 1: CTM-based requirement model example.

5 Weights and Potentials

Depending on the selected modeling approach, we see two major possibilities for the introduction of test selection criteria: the test case (condition) weights and the potential of dedicated test events.

A test case may be characterized by a sequence or collection of conditions (e.g. a sequence of nodes within a tree starting from the tree root “top” level to a “low” level terminating node) representing a test case. In this context tree edges between tree paths may have specific “weight” values (the sum of all edges from one node to all its lower level nodes is 100 %). In case of no weight assignment the equal distribution is assumed.

The overall condition weight of a single test is calculated by multiplying all single weight values corresponding to the test case edges of a single test (system component or path under test). Weight values may be retrieved from calculations or empirical observations as e.g. frequency of scenario occurrence or due to other empirical categories like fault risks, severity etc. the test may discover.

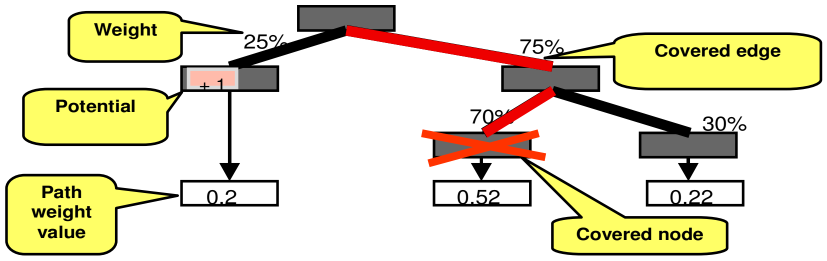


Fig. 2: Value assignments in classification trees.

In addition to the weights, a system feature (e.g. tree node) may have a potential that represents its specific importance. From the mathematical definition the potential x extends the weight by additional input condition. The potential is a mean to increase the test case weight values of all corresponding tests (e.g. subsequent tree paths).

Furthermore we introduce the qualification “covered” for those elements in the model which have been already used in a previous test, e.g. that part of the tree has been involved by an earlier test run and their portion in the tree that is not part of any other tests will be left in further calculations. The influence of the “covered” tree edge weights and node potential values has been considered by the algorithm in the following section.

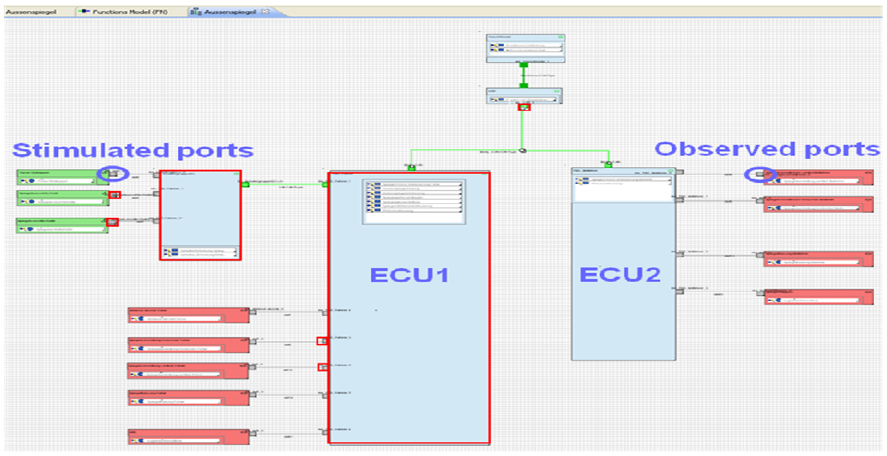


Fig. 3: Value assignments for system ports and components.

Other model approaches may use different model elements, e.g. ports and units. The understanding of weights as fixed representation of distributions and potentials as variable values (due to the setting by experts) will be used in all cases. In the PREEvision model [9] illustrated in Figure 3 stimulated sensors and observed reactors will been associated by weight values, system components may be attributed by potential values.

6 Generic Algorithm

Our algorithm for the calculation of test case priorities is based on the sum of weight and potential values that have been assigned to the conditions and events in a model that represents the set of test cases in a test campaign. In the following a generic algorithm has been defined.

Within the first step all weight values of all possible test (conditions) have to be calculated by multiplying all single weight values that belong to a single test in the model.

In a second step of the algorithm a factor will be calculated for each test that intends to reflect the relevance of a test case in addition to its test case weight. The following factor F identifies the portion of condition of a test case together with a consideration of the potential values assigned to test in comparison to the total sum of uncovered conditions and test potentials:

$$F = \frac{\#(\text{features in test}) - \#(\text{covered features in test}) + (\text{sum of potentials of test})}{\#(\text{all uncovered features in model}) + (\text{sum of all potentials in model})}$$

Next step is the multiplication of condition weight values and the factor F of each test case. The priorities of the test cases are according to the resulting calculated values: the test case with the highest value has the highest priority, etc..

Obviously the use of potentials is important for the priority of the test cases and may change the final ordering for the execution. The values may come from some external sources or are due to subjective experiences. In the latter case it may be difficult to find the “right” value for the importance of a node potential and any responsible person may wish to know about the influence of its value assignment before fixing the final decision.

Therefore we like to mention that it is possible to calculate the exact threshold value for a potential for balancing the priority of a particular test case in relation to any other test case. The calculation is based on the rule of three with one unknown variable x for a system feature potential. E.g. if someone wish preference of a test case to become the top test case the threshold value for its feature potential should be calculated in comparison to the current top test case of the whole model. The assignment of any potential value greater than x will change the resulting test case priority order according to this constraint.

Furthermore special consideration is required in case of combinations of multiple models, i.e. import of sub-models. Due to the modification of test case weights by the relevance factor F it is required to normalize the weight values of a sub-model calculation before they can be used in referring models.

7 Applications

The approach presented in the previous section has been used in different industrial domains with heterogeneous system and test models. In the following we refer to two samples: (1) the classification tree method applied in the telecommunication sector and (2) the newly developed architecture-driven test development approach initially used in the automotive domain.

7.1 Classification Tree Method

As the CTE modeling approach has been applied by Nokia Siemens Networks in a project at the systems test division. For that purpose, a section of the service logic for mobile telephony has been modeled using the classification tree method, yielding 31 test cases in the first step, as described in section 4. After applying some node weights and using the algorithm described in section 6, this number has been reduced to the six most important test cases that still cover all functional requirements.

Test runs using these six test cases have detected two faults in the service implementation. After execution of the remaining 25 test cases, no further faults have been found.

Table 1: Application in the telecom domain

	All test cases	Reduced # of test cases	Difference
# of test cases	31	6	25 (80%)
Requirements coverage	100 %	100 %	0
# of found faults	2	2	0

Thus, systematically reducing the number of test cases has resulted in a potential reduction of the testing efforts by more than a factor of five. At the same time, the coverage of all functional requirements is assured. Encouraged by this experience, Nokia Siemens Networks is currently introducing the classification tree method on a larger scale at systems testing. First results from other projects show similar results.

7.2 Architecture-Driven Test Development

The Architecture-driven test development method uses different architectural viewpoint for system models as introduced in [10]. Initially it has been defined in the context of an automotive case study while using the PREEvision software. This developed test derivation method [3] is useful to identify e.g. so-called PER-based tests, which consider sets of preconditions (P), events (E), and reactions (R). Basic PER validation tests are derived from the requirements available in the functional view of the system. Further tests can be derived due the inclusion of e.g. technical architecture information (specifying the hardware realisation of the SUT). The architecture model of the SUT (e.g. as illustrated in section 5) is used to assign and calculate the priorities for the derived test cases.

PER test cases can be described independent from particular notations but need to contain information related to precondition and event identifier weight values assigned to system component ports. A minimal test definition may be a text string that includes port identifier as e.g. “P1-E1-R1-R1.1”. Collected weight and potential values associated to these tests will be used to calculate the overall test case priority. Similar to the CTE related tests the algorithm from section 6 will be applied iteratively. A minimal example assuming an equal distribution for port values could lead to a test case ordering that follows the number of PER elements represented by the test, i.e.: priority(P1-E1-R1) > priority(P1-P2-E2-R2) > priority(P1-P2-P3-E3-R3) since more preconditions will decrease the probability of a test case.

8 Tool support

Tool support is depending on the test development method and involved system and test models. Existing tools need to be extended or new standalone tool are required.

The weighting algorithm has been integrated into the CTE XL tool introduced in section 4 for the case study reported in section 7.1. This extension to CTE XL is in use at Nokia Siemens Networks as an internal tool. It is being developed under an agreement between the CTE XL developers and Nokia Siemens Networks and is not publicly available. The following is dedicated to the CTE extension but also applicable for other system or test models.

When creating a model, the tool initially assigns the same weight to all alternative input conditions of a tree level in such a way that the sum of the weights is 100%, i.e., for n edges on one level, every edge is assigned a weight of 100%/n. Whenever an edge is added or removed, the weights are re-balanced to ensure that the sum remains at 100%.

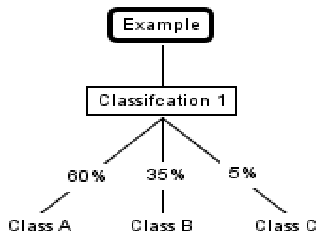
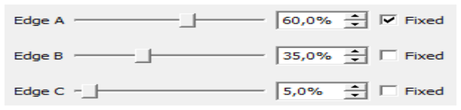


Fig. 4: Sample tool support for test weight input

The user can then adjust the weight of the individual edges by selecting the appropriate option in the context menu of the desired node. A window opens which allows changing the weights of all edges leading away from that node. In doing so, the weights of the other edges on the same tree level are changed proportionally to their former weight in order to keep the sum at 100%.

This makes it difficult to set a certain weight configuration when there are more than two edges on a level. For example, imagine a tree level with three edges A, B and C. When a user successively adjusts the weights of edge A and of edge B, the previously assigned weight of edge A is changed. Therefore, it is possible to declare the weights of certain edges as fixed. However, the weight of at least one edge on each tree level must remain dynamically assignable. In the example, the user would declare the weight of edge A as fixed. When changing the weight of edge B, only edge C's weight would be automatically adjusted, preserving the weight of edge A.



Edge A 60,0% [✓ Fixed]

Edge B 35,0% Fixed

Edge C 5,0% Fixed

Fig. 5: Adjusting edge weights

Additionally, it is possible to assign a fixed potential to each node, allowing the user to designate particularly important or unimportant test cases. Since there are no constraints concerning these potentials, they can be directly changed by the user. The initial potential is set to zero for all nodes.

After the test cases are generated, they can be sorted according to the resulting priority. The most important test cases are shown at the top of the list so they can be created and executed first. Moreover, the smallest subset of test cases covering all functional requirements can be determined automatically. This way, the minimum number of test cases required for testing all requirements covered by the modeled feature can be quickly determined.

An additional feature of the CTE XL extension is the dynamic reordering of test cases after the execution of a test case. This makes sense because after a test case has been executed, the nodes it covers have been tested. These nodes represent decisions in the service logic and, possibly, functional requirements. Therefore, it is more promising to execute a test case which covers different nodes next, since it is more likely that other faults are found by test cases which cover different aspects of the service logic. Since the preparation of test cases as well as the fixing of faults can take considerable amounts of time, it is worthwhile to try to find faults at the earliest possible chance.

This goal is achieved by marking a test case as executed in the model. The edges associated to this test case are then marked as covered, which reduces their weight by a configurable amount. This can result in a change of the order in which the test cases should be executed, depending on the concrete model and its weights and potentials. In order to keep track of the test cases that are marked as executed, they are labeled as such and they remain at a fixed position inside the test case list when the list is reordered due to the weight changes.

9 Conclusions

In our approach that is applicable on models like classification trees or system architecture we have introduced the terminology, method and algorithm to introduce weight and potential values for the prioritization of test cases. The approach allows finding an order for test case execution and first results using an extended implementation of e.g. the classification tree editor CTE XL have shown that most important test cases lead to an early discovering of faults in the main modules.

As a consequent we request an indication of weight/relevance information as part of requirement specification as well as a documentation (fault index) of experiences from earlier test runs to have a means to identify critical parts in the classification tree (i.e. critical modules of the system under test) with higher weight values and therefore greater priority.

Our future work will address the impact of algorithm known e.g. from Google PageRank and further experiments from simulations.

References

[1] S. Alekseev et al.: Systematic Approach for using the Classification Tree Method for Testing Complex Software-Systems. In Proceeding of the IASTED Conference on Software Engineering, Innsbruck, Austria, February 2007.

[2] S. Alekseev et al.: Reuse of Classification Tree Models for Complex Software Projects. CONQUEST 2007, Potsdam, Germany, September 2007.

[3] G. Din, K.-D. Engel: An Approach for Test Derivation from System Architecture Models applied to Embedded Systems. MoTiP'09 June. 2009, Enschede (NL).

[4] M. Grindal et al.: An evaluation of combination strategies for test case selection. In: Springer Science + Business Media, LLC 2006.

[5] M. Grochtmann, J. Wegener: Graph Theory in the Control Flow Analysis of the large time critical Aplications. In Proceedings of the 8th International Software Quality Week, San Francisco, USA, May 1995.

[6] A. Hoffmann et al.: Application of the Classification Tree Method for Test Modeling in Complex Software Projects, 4th World Congress for Software Quality (4WCSQ), Bethesda, Maryland (USA), September 2008.

[7] E. Lehmann, J. Wegener: Test Case Design by Means of the CTE XL. In Proceedings of the 8th European International Conference on Software Testing, Analysis & Review, Kopenhagen, Denmark, EuroSTAR 2000, December 2000.

[8] J-M. Kim, A. Porter: A History-Based Test Prioritization Technique for Regression Testing in Resource Constrained Environments. ICSE'02, May 19-25, 2002, Orlando, Florida, USA.

[9] PREEvision: Aquintos software tool. http://www.aquintos.eu/?getlang=en.

[10] T. Ringler et al.: An Approach to Tool-based Design of Electrics/Electronics Architectures. In: ATZe worldwide Edition: 2008-01.

[11] J. Wegener et al..: Tool-Supported Test Case Design for Black Box Testing by Means of the Classification- Tree Editor. In Proceedings of the 1st European Interna-tional Conference on Software Testing Analysis & Review, London, Great Britain, pages 169 – 176. EuroSTAR, 1993.