Weighting Influence of User Behavior in Software Validation

A. Bertolino
ISTI-CNR
via Moruzzi, 1
56124 Pisa, Italy
antonia.bertolino@isti.cnr.it

E. Cartaxo
GMF-UFCG
Av. Aprígio Veloso, 882
Campina Grande, Brazil
emanuela@dsc.ufcg.edu.br

P. Machado
GMF-UFCG
Av. Aprígio Veloso, 882
Campina Grande, Brazil
patricia@dsc.ufcg.edu.br

E. Marchetti
ISTI-CNR
via Moruzzi, 1
56124 Pisa, Italy
eda.marchetti@isti.cnr.it

Abstract

Validation is an essential part of software development, and testing is a practical and widely used approach. The emerging methodology is model-based testing, in which test cases are derived from a model of software behaviour. In this paper we claim that also user behaviour should be taken into account for test planning purposes. We introduce a pragmatic approach called WSA, which derives the test cases from a state-based model, while also accounting for weights that consider relevance wrt user behaviour.

1 Introduction

For decades a large effort of research in software engineering has focused on usability, maintainability, reliability and other characteristic "ilities" of software applications. With software increasingly pervading our lives, guaranteeing adequate levels of quality and dependability becomes paramount. To this purpose, novel development paradigms and processes are incessantly proposed. An essential component of the software development lifecycle is testing [2], which is a means applicable for both evaluating product quality and improving it, by identifying defects and problems. Testing amounts to a systematic observation of the software in execution to assess its properties.

The emerging approach today is the so-called Model-based Testing (MBT), in which a suite of test cases is derived, possibly in automated way, from a model that represent the system under test (SUT). Many different techniques have been developed, differing for the aspect of the system that is captured in the model, for example state-based, scenario-based, data-flow based test generation could be applied. While MBT has proved to be cost-effective in many cases, a limitation is that model-based testing is neutral with respect to end-user behavior. The test cases are derived from a model of the software system, separated from user's profiles, and therefore the validation will tend to consider equally all the components within the model, irrespective of how the software will be actually used.

In the eighties, a different approach to software engineering has been introduced, pioneered by Bohem [5], called the "Value-Based Software Engineering" (VBSE). The leading principle of VBSE is that not all failures are equal, and therefore all activities of software development, including testing, should be valued according to their cost. This is certainly true for test case generation, which takes an extensive part of software development effort. We compare the two approaches (MBT and VBSE): each brings advantages, but at the same time each also ignore crucial aspects of the software. In this paper, we realize a combined software testing philosophy which consists into taking the model as a reference to ensure that all program components are adequately taken into account, as is done in MBT. In particular, we adopt the similarity based on identical parts of the model, but then we annotate such model with weights that value end-user behaviour. Hence, the contribution of this paper to CUB2008 consists into an original approach for test planning and generation, called the Weighted Similarity Approach (WSA), which weights the influence of user behaviour in software validation. In presenting the approach, we also provide an overview of the main concepts of the test process. The paper is structured as follows. In Section 2, an overview of the test process considered in this work is presented. In Section 3 the main concepts or value based software engineering are highlighted. In Section 4, our approach is introduced. An example of the application of our approach is presented in Section 5. Related works are presented in Section 6. Conclusions and future plans are drawn in Section 7.

2 Test Process Overview

Software testing is an integrated and significant activity during the whole software life cycle. Good practice suggests to account for test planning and management since the early stages of software development, during requirements analysis, and to proceed with its organization and refinement systematically and continuously during the entire development process down to the code level.

The process regulating the testing activities can vary and depends strictly on the adopted development lifecycle model; anyhow the main phases of a test process can be resumed in the following stages [3]:

- **Planning:** As for any other process activity, the testing must be planned and scheduled. Thus the time and effort needed for performing and completing software testing must be established in advance during the early stages of development.
- **Test cases generation:** According to the test plan constraints a set(s) of the test cases must be generated by using a (several) test strategy (ies).
- **Test cases execution:** Test cases execution may involve testing engineers, outside personnel or even customers.
- **Test results analysis:** The collected testing results must be evaluated to determine whether the test was successful (the system performs as expected, or there are no major unexpected outcomes) and used for deriving measures and values of interest.
- **Problem reporting:** A test log documents the testing activity performed. Anomalies or unexpected behaviours should be also reported.
- **Post-closure activities:** the information relative to failures or defects discovered during testing execution are used for evaluating the performance and the effectiveness of the developed testing strategy(ies) and determining whether the process development adopted needs some improvements.

Among the above mentioned phases this paper focuses mainly on the first two: test planning and test case generation. We pay particular attention to strategies for selecting the parts (functionalities) of the software products on which the testing must concentrate in order to avoid loss of time and effort. Generally this is a crucial aspect for software developers, which is often left to the intuition or expertise of the program managers or testers. Unfortunately wrong decisions in this contest can considerably increase the overall effort and time required for delivering a "good" product.

Here the focus of our proposal is to guide to appropriate testing choices since the first phases of development. To find applicable solutions, we adopted two important guidelines: maximizing usability and automation.

Concerning usability, what we propose is an easy-to-use and ready-to-apply approach, which minimizes as much as possible the required additional formalism or ad-hoc effort specific for testing purposes. These aspects are immediately translated into an increase in the cost of software testing, which is improbably accepted.

The second constraint considered is automation. The increasingly strict delivery time imposed by markets forces software developers to accelerate product development as much as possible. This often is translated into reducing the time devoted to for performing software testing, which is one of the most expensive activities of the development. Consequently the testing phase is partially skipped and the software products released in advance only because there is not enough time for testing them properly. One of the ways to pull down the overall testing time is to considerably increase as far as possible the automation in test cases derivation, execution and validation, thus reducing the manual labour.

Our contribution concerns the definition of an approach which supports the user both in the choice of the most important software elements, based on the foreseen user behaviour, on which the testing effort must be concentrated, and in the automatic generation of the appropriate test cases by using the available product specification.

3 Value Based approach

Value based software engineering has been introduced in 1981 with Boehm's Software Engineering Economics book [5] and has inspired the value based management movement in the early 1990 [4]. Its philosophy, is that "quality should not be a goal in itself in the absence of favorable economics". Since then, consideration for software-related value has expanded its scope and values have been incorporated deeply into successful developments process. The common purpose has been promoting the different considerations/measures/knowledges to the foreground so that the software engineering decisions could be guided and optimized by these values. Saying exactly once for all what "value" represents in this discipline is not possible. The referred numbers can be related to various topics. They can represent the economical and financial aspect, they can be percentage or probability, they can represent abstract concepts as quality, availability, usability and so on.

In our approach values are the parameters or choices useful to adapt the test strategy and the test cases derivation to specific user needs and expectations. Generally the various system functionalities do not have the same "importance" for overall system performance or dependability, and the testing effort should be planned and scheduled accordingly. Different criteria can be adopted in order to define what "importance" means for test purposes, e.g., component complexity, or usage frequencies (such as in reliability testing [9]).

Often, these criteria are not documented or even explicitly recognized, but their use is implicitly left to the sensibility and expertise of the managers. The basic idea in our approach is that we ask test managers to make explicit these criteria, and we provide them with a systematic strategy in order to use such information for test planning. The main task is express for each functionality a value, belonging to the [0,1] interval, representing its relative "importance" with respect to the other functionalities. This value, called the weight, must be assigned in such a manner that the sum of the weights associated to all the children of one level is equal to 1; the more critical a functionality the greater its weight.

Several criteria for assigning the importance factors could be adopted. Obviously this aspect in the proposed approach remains highly subjective, more in the realm of expert judgment than mechanisable methods, but here we are not going to provide a quick recipe on how numbers should be assigned. We only suggest expressing in quantitative terms the intuitions and information about the peculiarity and importance of the different part of the system to be developed, considering that such weights will correspondingly affect the testing stage. In our intuition, that only empirical evidence from the strategy usage history can legitimate, we believe that the strategy should be robust enough to moderate deviations. However it is worth noting that the process of functionalities annotation implies a beneficial side-effect: for assigning the appropriate values, the manager are forced to reflect on the relative complexity of each functionality with respect to the context in which it is inserted. Consequently, they pay attention to the parts where problems could be more critical and become conscious of the importance of each node for the system development.

4 Weighted Similarity Approach

The main goal of this approach is to exploit the software engineering knowledge and experience to minimise the size of a test suite by keeping in it only the test cases that can be feasibly executed according to user behavior and resources available to the testing process. For this, we introduce the concept of *similarity* and we show how it can be used for test cases selection and generation. In particular, following the usability principle, the WSA approach is based on use case templates derived by applying a Model Based Testing (MBT) approach.

By observing a generated test suite, it is usually easy to detect a considerable number of redundant test cases [8]. Since they usually cover the same set of functionalities, these similar test cases can be excluded from the test suite, without compromising, for example, functionality coverage. Moreover, two test cases may be so similar that they only differ by a single execution step.

For handling redundancy, we define a systematic method for measuring the *similarity* among test cases. This value can be exploited for minimizing the test suite, i.e. keeping in it only the most different test cases for execution. For measuring the *similarity*, we consider the number of identical actions (of user or system) for each pair of test cases. For each identical action, one unit is added to the similarity value of that pair.

The details of this method are presented in [6]. Here we only summarize the main steps. In particular, to maximize coverage, the biggest test case is always chosen when one of two similar test cases need to be discarded. When they have the same size, one is randomly chosen.

The drawback of this strategy is that important test cases could be eliminated. For instance, a test case may be focused either on a frequent use (sub)-functionality, or on a very important functionality with respect to user needs or on critical path of the application. Since this strategy does not distinguish among the importance of test cases, crucial ones may be discarded. For this we integrate similarity method with a value based test strategy, thus the Weighted Similarity Approach. WSA foresees the test cases selection by prioritizing accordingly with the values set by the end user to each functionalities. The idea is that when choosing between two similar test cases to be discarded, the one that has a greater value is kept. In the original strategy, a random choice is performed when they have the same size. Therefore, we aim at testing suites that have the most different test cases and yet these are also the most important ones.

WSA uses the same use case document developed during requirement phase and provides facilities to the end user for setting the proper values to the different functionalities. From the use cases, a set of test cases are selected each one representing a flow in the use case. Even if the approach is not strictly related to a specific model based approach, in this paper for aim of simplicity refer to the representation presented by Nogueira and his colleagues in [10]. Figure 1 is an example.

Using such template the user can set the desired path coverage (i.e. amount of desired test cases) and provide the weights to be associated to each possible flow of the use case. Every flow represents one or more different test cases. In this paper, we consider that each weight indicates the expected frequency of use for a given flow. In summary, WSA approach foresees that user specifies the use cases by filling the templates and they assign weights to each flow specified in the template by taking in consideration that the sum of weighs for each flow of a branch is 1.

Giving the use case specification with weights assigned, test cases similarity and *final weight* for each flow is computed. Similarity between two test cases is computed as the number of common steps in the two test cases. The *final weights* of a test case is obtained by multiplying the attributed weight in their branches. As result, a weighted similarity matrix is built with test cases as columns and rows, where each element is defined as the similarity between two test cases divided by the weight of the test case of the row. Note that, the most similar test cases must be eliminated and the most important must be kept. For this, we balance the similarity with the weight of each test case.

In order to choose a test case to be removed, we look for the highest value in the matrix. This corresponds to a test case that is very similar to other test case and has a lower weight. If there is a tie, then the smallest test case must be eliminated. If there is a tie again, random choice is applied.

5 Example: A Phonebook agenda

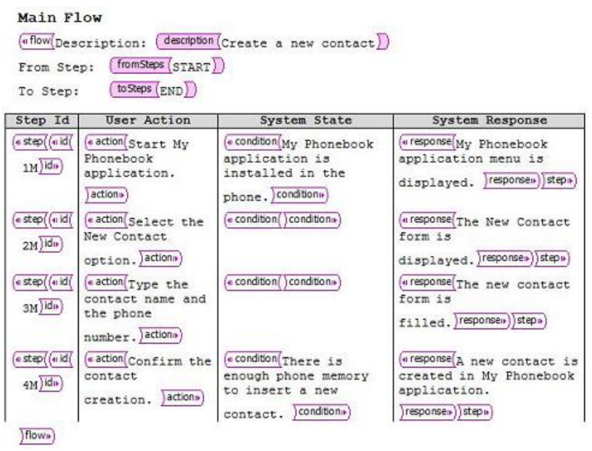
This section presents a simple example to illustrate the proposed strategy. In this example, we illustrate how use cases are specified by using the template defined by the Target tool [10]. Then the results obtained by applying the original and WSA are presented and discussed (Section 5.1 and 5.2 respectively).

The use case describes the creation of a new contact in the Contact list. In Figure 1, the main flow of this use case is presented. This flow represents an user that add a contact with success.

Figure 2 presents two alternative flows. The first one describes the scenario where the user can cancel the creation of the new contact. This happens in two cases: the form is opened (Step ID 2M) and the form is filled (Step ID 3M). The second flow describes the scenario where the new contact is not created because there is not available phone memory, this can happen when the user try to add the contact (after filling the form, that is, Step ID 3M).

By following the Step IDs, a graphical representation is automatically derived and used for test case selection. This model can also be used by test designers as an alternate view of the specified behaviour (this can be seen in Figure 3).

From Figure 3, we can see that after the user executes Step 2M, they can either execute Steps 3M or 1B. Also, from Step 3M, the user can opt for Steps 4M, 1C or 1B.



Main Flow

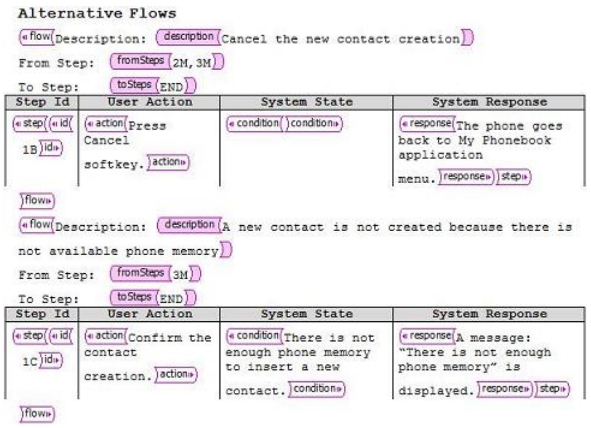
Description: Create a new contact

From Step: START

To Step: END

Step Id	User Action	System State	System Response
1M	Start My Phonebook application.	My Phonebook application is installed in the phone.	My Phonebook application menu is displayed.
2M	Select the New Contact option.		The New Contact form is displayed.
3M	Type the contact name and the phone number.		The new contact form is filled.
4M	Confirm the contact creation.	There is enough phone memory to insert a new contact.	A new contact is created in My Phonebook application.

Figure 1. Creating a New Contact - Main Flow



Alternative Flows

Description: Cancel the new contact creation

From Step: 2M, 3M

To Step: END

Step Id	User Action	System State	System Response
1B	Press Cancel softkey.		The phone goes back to My Phonebook application menu.

Description: A new contact is not created because there is not available phone memory

From Step: 3M

To Step: END

Step Id	User Action	System State	System Response
1C	Confirm the contact creation.	There is not enough phone memory to insert a new contact.	A message: "There is not enough phone memory" is displayed.

Figure 2. Creating a New Contact - Alternative Flows

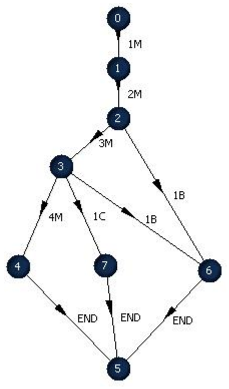
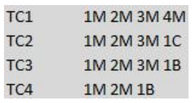


Figure 3. Graphical Representation of the Behavior Model

From the graphical representation 3 , we can obtain 4 test cases (Figure 4)



TC1	1M 2M 3M 4M
TC2	1M 2M 3M 1C
TC3	1M 2M 3M 1B
TC4	1M 2M 1B

Figure 4. Test Cases

In the rest of this section we compared results obtained by applying Similarity and WSA. In both cases, we considered that we had resources to execute only 50% of test cases, i.e., only 2 out of 4.

5.1 Similarity Strategy - Results

From Figure 4, we have three very similar test cases: TC1, TC2 and TC3. By applying the original similarity strategy, two of these test cases are chosen to be removed whereas TC4 is kept (since it is the less similar). The choice between the similar test cases is random as they have the same size. In this case, let's assume that the final set is composed of TC3 and TC4, by removing TC1 and TC2.

5.2 Weighted Similarity Approach - Results

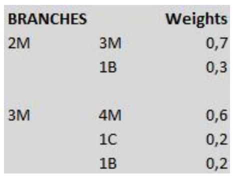
Now, let's consider the weighted similarity approach. For this, we define weights for each branch that is originated by alternate flows after step 2M and 3M (Figure 5). In this case, this weight is related to the expected frequency of use. The goal is to keep test cases that allow for the best possible coverage of flows and steps that are going to be more frequently used in practice, making sure they are tested at this level.

By considering the weights, TC1 is now considered an important test case because it traverses steps 3M and 4M that have higher weights. So, this is now distinguishable from TC2 and TC3. Consequently, TC2 and TC3 are going to be removed.

Note that the main difference from the two approaches is that, by following the original one, we may discarded a very important test case to the user (TC1) depending on the random choice, whereas by the weighted strategy TC1 is always kept, guaranteeing that this important functionality will be fully tested.

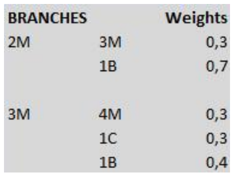
Now, suppose that the new weights are showed in Figure 6. TC4 is the most important and less similar, so this must be kept in the test suite. TC1 and TC2 are the less important and are the most similar. Then, we keep TC3 and TC4 in the test suite.

See that, we have applied different weights (Figures 5 and 6) and obtained the best results.



BRANCHES		Weights
2M	3M	0,7
	1B	0,3
3M	4M	0,6
	1C	0,2
	1B	0,2

Figure 5. Weights



BRANCHES		Weights
2M	3M	0,3
	1B	0,7
3M	4M	0,3
	1C	0,3
	1B	0,4

Figure 6. Weights

6 Related Works

Model based Testing is an emerging approach for designing, specifying and deriving test cases and many approaches. Different techniques and approaches have been developed for covering the various aspect of software testing. We refer to [11] for a more complete overview of the most recent proposals. In this section, we only mention the approaches that mostly inspired the WSA proposal which combines Model Based techniques with Value Based approaches. In particular, concerning the former, we mainly focused on approaches for test suite reduction.

Usually available works for reducing the size of test suite are focused on 100% coverage criteria and does not take in consideration resources availability for scheduling and performing the test phase. For space limitation among them we only mention the proposal of Harrold et al. [7], which is based on a requirements traceability matrix. For each requirement a the test cases is defined. The aim is to reduce the test suite by guaranteeing that 100% coverage of requirements is preserved. The WSA approach aims to extend this view by join test reduction methodologies with values managements. For this WSA exploits the advantages of the similarity strategy, presented in [6] for test reduction. Similarity guarantees that when resources (time an effort) have to be properly distributed for test cases execution, only the different test cases are kept in test suite.

Considering the value based approach different proposals are in literature (see [4] as a reference). Among those that inspired the WSA approach one of the most widespread is the Musa's Software-Reliability-Engineered Testing (SRET) proposal [9]. Faults can be more or less disturbing depending on whether, they will eventually show up to the final user. Software testing should be focused on the *reliability*, i.e. the probability that the software will execute without failure in a given environment for a given period of time. For this the values representing how frequently inputs that cause a failure will be exercised by the final users are estimated. These are exploited for selecting the test cases in such a way to approximate as closely as possible the future usage in operation. The other very WSA close work is the Cow_Suite (COWtest pluS UIT Environment) [1]. This is an integrated and automatic approach for generating and planning a suitable set of test cases, starting with the UML documentation. This methodology combines a value based test strategy, called Cowtest and a method for test cases generation, called UIT. In particular Cowtest accordingly with the values assignments provides two different test planning schemes: testing must respect a certain resource investment, which in practice we translate into fixing the number of test cases; or the test cases must cover a fixed percentage of functionalities.

WSA try to integrate all the above mentioned methodologies into a unique, usable environment, which can be useful both for planning the testing phase and for suitable selecting the test case to be executed.

7 Conclusions

We have argued that user's behaviour, in particular which components they will exercise mostly, or which functionalities maybe more critical, should be taken into account in test planning and generation. We propose that weights of user's related importance or criticality are introduced into Model-based Testing, which only considers software behaviour models. We have introduced the WSA approach, that performs test reduction by discarding redundant test cases that are similar to retained test cases, but keeps important ones. WSA put together two important concepts: the similarity and the importance of test cases. The similarity is calculated by observing the number of identical steps between each pair of test cases. The importance of test case is defined by WSA's user when he applies the weights to branches. Then, by applying the WSA approach, we obtained a reduced test suite. The number of test cases in final test suite is defined by WSA's user by observing the available resources. The test cases in the final test suite are the less similar and the most important. A simple example has been developed, for the addition of contacts into a Phonebook agenda. A crucial part of the approach is how to decide the weights. For this, future work will need to integrate value-based approaches and user profiling methodologies. Further validation is also planned.

References

[1] F. Basanieri, A. Bertolino, and E. Marchetti. The cow_suite approach to planning and deriving test suites in uml projects. In *Proc. Fifth International Conference on the Unified Modeling Language - the Language and its applications UML 2002*, pages 383–397, Dresden, Germany, 2002.

[2] A. Bertolino. Software testing research: Achievements, challenges, dreams. In *FOSE '07: 2007 Future of Software Engineering*, pages 85–103, Washington, DC, USA, 2007. IEEE Computer Society.

[3] A. Bertolino and E. Marchetti. Software testing (chapt. 5). *Guide to the Software Engineering Body of Knowledge SWEBOK*, pages 5–1, 2004.

[4] S. Biffl, A. Aurum, B. Boehm, H. Erdogmus, and P. Grünbacher. *Value-Based Software Engineering*. Springer-Verlag New York, Inc. Secaucus, NJ, USA, 2005.

[5] B. Boehm. *Software Engineering Economics*. Prentice Hall PTR Upper Saddle River, NJ, USA, 1981.

[6] E. G. Cartaxo, F. G. O. Neto, and P. D. L. Machado. Automated test case selection based on a similarityfunction. In *MOTES07 - Model-based Testing - Workshop in conjunction with the 37th Annual Congress of the Gesellschaft fuer Informatik*, volume 110, pages 381–386. Lecture Notes in Informatics (LNI), September 2007.

[7] M. J. Harrold, R. Gupta, and M. L. Soffa. A methodology for controlling the size of a test suite. *ACM Trans. Softw. Eng. Methodol.*, 2(3):270–285, 1993.

[8] P. C. Jorgensen. *Software Testing: A Craftsman's Approach*. CRC Press, Inc., Boca Raton, FL, USA, 1995.

[9] J. D. Musa. Software-reliability-engineered testing. *Computer*, 29(11):61–68, 1996.

[10] S. C. Nogueira, E. G. Cartaxo, D. G. Torres, E. H. S. Aranha, and R. Marques. Model based test generation: A case study. In *Workshop on Systematic and Automated Software Testing*. Workshop on Systematic and Automated Software Testing, October 2007.

[11] M. Utting and B. Legeard. Practical Model-Based Testing. *Morgan Kaufmann*, 1(1.4):1–5, 2007.