

INPUT PARAMETER MODELING FOR COMBINATION STRATEGIES

Mats Grindal
School of Humanities and Informatics
University of Skövde
Sweden
email: mats.grindal@his.se

Jeff Offutt
Information and Software Engineering
George Mason University
Fairfax, VA 22030, USA
email: offutt@ise.gmu.edu

ABSTRACT

Combination strategies are test methods that generate test cases based on input parameter models. This paper suggests a structured modeling method used to translate requirements expressed in a general format into an input parameter model suitable for combination strategies.

This paper also describes results from two initial experiments exploring the efficiency and effectiveness of the modeling method. These results indicate that the resulting models may contain enough information to detect the vast majority of faults in the system under test. Further, results indicate that the modeling method is simple enough to use in practical testing.

KEY WORDS

Software Testing, Model-based Testing, Combinatorial Testing, Category Partition

1 Introduction

In testing, the system under test (SUT) is executed by feeding selected inputs to the system. The behavior is observed and compared with some expected behavior to determine whether or not the system behaves correctly.

The input space of a SUT can often be expressed in terms of parameters, each with a number of possible values. Every complete valid combination of parameter values is a potential test case. With increasing numbers of parameters and parameter values, the number of possible combinations soon becomes infeasible for testing.

Combination strategies are test case selection methods designed to handle the combinatorial explosion. The general idea is to identify a manageable subset of test cases based on some coverage criterion. More than 15 combination strategies have been surveyed and compared by Grindal et al. [8, 7]. The characteristic features of a combination strategy are the coverage criteria supported and the algorithm used to identify a suitable subset. As with most test case selection methods, the level of coverage greatly impacts the size of the final test suite [7, 8].

Figure 1 shows a test process for combination strategies. In step 1 the tester selects one or more combination strategies to use. Step 2 is to formulate an input parameter model. An *input parameter model* (IPM) is an abstract representation of the input space of the system under test. It is

used as input to the combination strategy in step 3, in which abstract test cases are generated. Step 4 gives the tester an opportunity to evaluate the abstract test cases, for instance with respect to size of test suite. When a set of abstract test cases has been judged to be adequate, the abstract test cases are translated into actual test cases in step 5. These are then executed in step 6 and the results are evaluated in step 7.

Step 2, input parameter modeling, is the focus of this paper. In its most basic form, the IPM lists a set of *parameters* (also known as categories [12]), each with a unique identifier. Each parameter has a number of associated *values*, also known as choices [12]. Each parameter value represents a non-empty partition of the input space.

It is often difficult to create IPMs. Informal specifications can be interpreted in various ways. There are significant design decisions and tradeoffs to representing aspects of the system. Grochtman and Grimm [9] concluded that identifying parameters and parameter values (categories and choices in their terminology) is a creative process that can never be fully automated.

Different testers will come up with different models, depending on creativity and experience. This also creates a potential for errors. Chen et al. empirically identified common mistakes that testers made during input parameter modeling [2].

A structured method or checklist to support input parameter modeling can decrease mistakes and increase the quality of the IPM. To our knowledge no such method or checklist exists for combination strategies. However, several suggested test methods exist that are related to combination strategies, including Category Partition [12] and Classification Trees [9].

This paper presents a basic eight-step process that is custom-designed to be used with combination strategies. Some initial experience using this process is also included in this paper. The basis for the formulation of the input parameter modeling method is the set of requirements on the contents of the IPM imposed by combination strategies. Section 2 describes these requirements. Section 3 provides an overview of the IPM process and gives details of each of the eight steps. The IPM process has been evaluated in a small experiment. The results of this experiment are presented in section 4. In section 5 our IPM process is compared to other similar input parameter modeling methods. Sections 6 and 7 summarize the paper and discuss

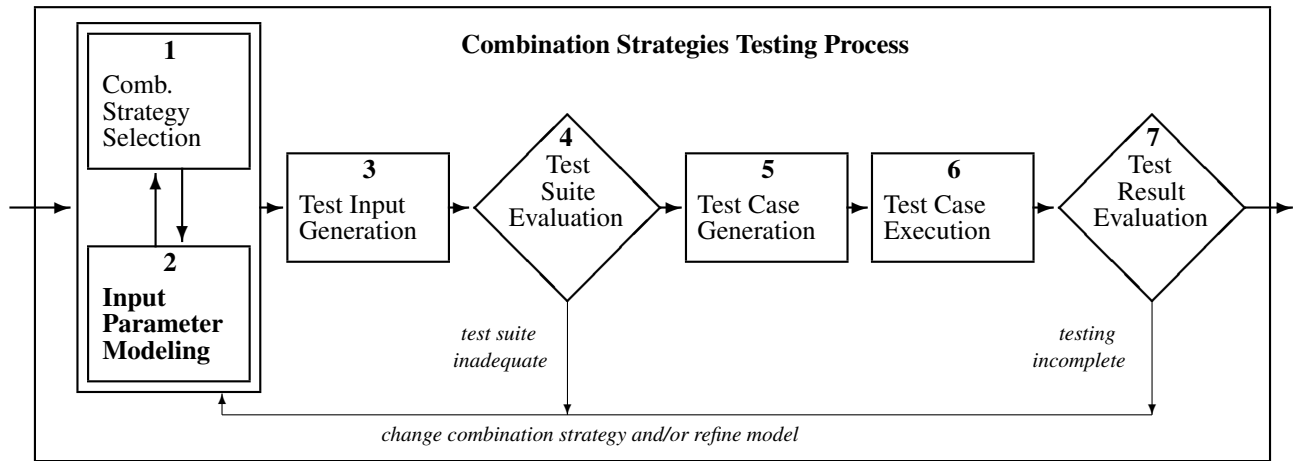


Figure 1. A Combination Strategies Testing Process. Step 2 - Input Parameter Modeling is Highlighted in On the Left.

future work.

2 Requirements on the Contents of the IPM

The basic functionality of combination strategies is to identify test cases by combining values of the different input parameters based on some combinatorial strategy [8]. The input to combination strategies is a *model*, which should enumerate parameters and parameter values. These represent the input space of the SUT.

Some sub-combinations of parameter values may be invalid. The input parameter model must contain enough information to prevent the selection of invalid sub-combinations. Some representation of these restrictions must be available in the model.

In the fifth step in the test process in figure 1, abstract test cases are translated into executable test cases. This translation has two parts. The parameter values should be converted to actual input values and an expected result should be derived. The first part is relatively simple to automate, but requires a *translation table* that contains mappings between the parameter values and values of the actual interface. The second part is harder to automate, as it requires the specification to be expressed in a semantically exact way. This is outside the scope of this study.

Often, there is a set of predefined test cases or the tester, based on her experience, wants some test cases to be included in the final test suite. Some combination strategies, for instance the AETG combination strategy [4] support the inclusion of preselected test cases. Hence, the input parameter model needs to support the specification of preselected test cases.

3 A Process for Input Parameter Modeling

Figure 2 shows a graphical representation of our input parameter modeling method. The eight steps of the method are described separately in the following subsections. The

numbering of the subsections correspond to the numbering of the steps.

3.1 Determine the Modeling Approach

Test case selection methods are generally categorized as white-box methods or black-box methods. In *white-box* methods, expected results in the test cases are identified from the specification but inputs are derived from the implementation. In *black-box* methods, both input and expected results in the test cases are identified from some version of the specification. This classification inspires two different approaches to input parameter modeling, *interface based* and *functionality based*. For each IPM, the tester chooses one of these approaches. The following subsections describe these approaches and discuss their strengths and weaknesses.

3.1.1 Interface-Based Input Parameter Modeling

The interface-based approach has been used in most of the previous research [8]. The idea is to use a one-to-one mapping between the interface parameters and the parameters of the IPM.

The obvious strength of this approach is that the identification of parameters is straightforward. The one-to-one mapping between the IPM parameters and the interface parameters also simplifies the translation into real test cases in the test generation step of the test process of figure 1.

A weakness of this approach is that not all requirements will be reflected in the interface. The effect of this is that the final IPM may be incomplete. Hence, additional values may be missed.

Another weakness is that some parts of the functionality may depend on combinations of specific values of several interface parameters. In the interface-based approach each parameter is analyzed in isolation with the effect that important sub-combinations may be missed.

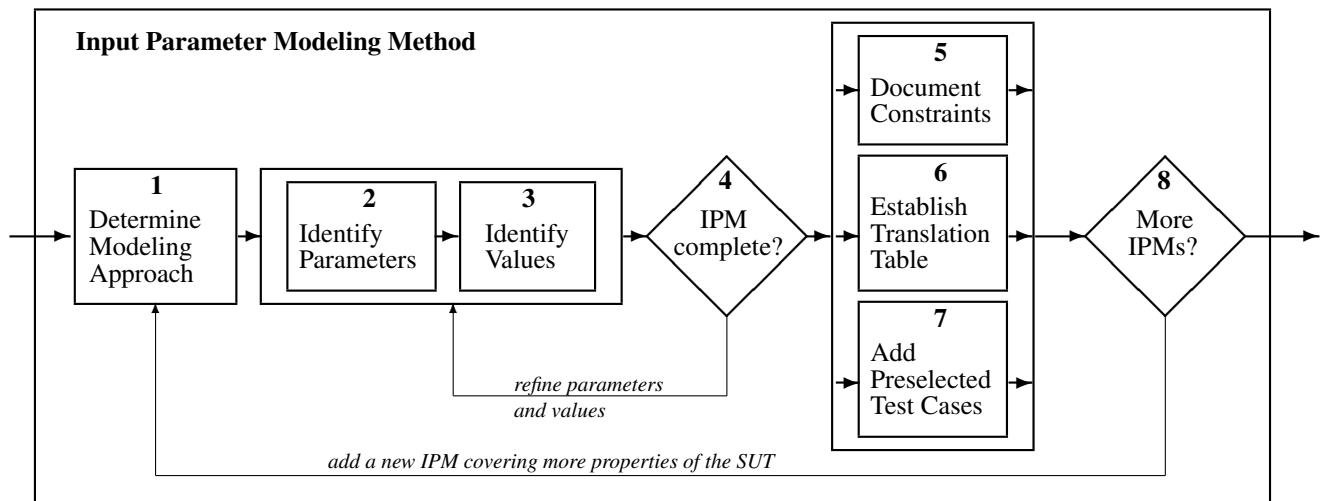


Figure 2. An input parameter modeling method.

A third weakness with the interface-based approach relates to the quality of the test cases. In general, it is good if actual usage is reflected in the test cases, which is easier to accomplish with a functionality based approach.

3.1.2 Functionality Based Input Parameter Modeling

Both Cohen et al. [5] and Yin et al. [13] suggest functionality oriented approaches to input parameter modeling. Functionality oriented input parameter modeling was also implicitly used by Grindal et al. [7]. The idea of the functionality based approach is to identify parameters and values that correspond to the intended functionality of the system under test rather than using the actual interface.

Some previous research suggest that a functionality based approach results in better test cases than the interface-based approach [5, 13]. There is no firm evidence to support this claim due to the lack of empirical data. However, there is an analytical motivation for this assertion. Compared with an interface-based approach, the functionality based approach should result in input parameter models that include more semantic information. If this is true, this is probably the biggest advantage with the functionality based approach. Transferring more semantic information from the specification to the IPM makes it more likely to generate expected results for the test cases, an important goal.

Another important strength of the functionality based approach is that the requirements are often available prior to the actual implementation. This means that the functionality based input parameter modeling, and thus the test case generation, can be started early in the development process.

In the functionality approach, identifying parameters and values may be far from trivial. Complicating factors are the size and complexity of the system under test and the informality and incompleteness of many specifications. This is a disadvantage of the functionality approach com-

pared to the interface-based approach.

Another disadvantage with the functionality approach turns up later in the combination test process of figure 1. In the fifth step of the process, “test generation,” abstract test cases are transformed into actual test cases. With a functionality approach the parameters of the IPM do not necessarily map one-to-one onto the parameters of the actual interface. Without a one-to-one mapping, the translation into actual test cases becomes more complex since values of two different IPM parameters may affect the same parameter in the actual interface.

3.2 Identify Parameters

Identification of parameters is closely connected to the selected modeling approach. When the functionality based modeling is used, parameter identification starts with an analysis of the specification or any other description of the behavior of the SUT. Different aspects of the functionality are identified and formulated as parameters. As far as possible, parameters should be independent of each other. The following checklist describe additional properties that a set of parameters should satisfy.

- **Missing factors** - Check whether there exists any factor that may impact the execution but does not have any associated IPM parameter.
- **Overlap** - If two parameters contain overlapping values they may be combined to one parameter.
- **Irrelevant parameters** - A parameter is irrelevant if none of its choices are included in any complete combination.
- **Number of parameters** - Based purely on cost, it is usually better to have many parameters with few choices than the reverse.

In the interfaced based parameter approach, each parameter in the interface of the SUT results in a separate IPM parameter.

The two approaches may result in different IPM parameters. To illustrate this difference consider testing the following function: `boolean find_element(list, element)`, which returns true if the `element` is found in the `list`.

If the functionality based approach is used, the IPM parameters may include A: length of list, B: contents of list, C: element occurrence. In contrast, the interface-based approach results in exactly two IPM parameters: A: list, B: element.

3.3 Identify Values

The value domain of each identified parameter is partitioned into groups of values. A key issue in any partition approach is how partitions should be identified and how representative values should be selected from each group. Equivalence Partitioning (EP) [11] and Boundary Value Analysis (BVA) [11] may be used for this purpose. The tester is free to make her choice based on available information and experience.

However, some strategies for identifying values are:

- **Valid values** - Include at least one group of valid values.
- **Sub-partition** - A range of valid values can often be partitioned into sub-partitions, such that each sub-partition exercises a somewhat different part of the functionality.
- **Boundaries** - Values at or close to boundaries are over-represented in defects.
- **Normal use** - If the operational profile focuses heavily on “normal use,” the failure rate is more dependent on values that are not boundary conditions.
- **Invalid values** - Include at least one group of invalid values
- **Balance** - From a cost perspective, it may be cheap or even free to add more partitions to parameters with fewer partitions than the rest.
- **Invalid partitions** - For each identified partition, check that it contains at least one element that is unique to that partition.
- **Missing partitions** - Check that the union of all partitions of a parameter completely covers the input space of that parameter.
- **Overlapping partitions** - Check that no element belongs to more than one partition.

3.4 IPM completeness decision

Both identification of parameters and identification of values include completeness checks. As identification of parameters and values is an iterative process, it may be difficult to determine the completeness until all parameters have been identified and partitioned. Thus, the tester must carry

out an explicit completeness check. the main purpose of this check is to make sure that the identified parameter partitions cover the complete input space of the SUT.

Figure 2 indicates the possibility of using more than one IPM to represent a SUT. In such cases, the completeness check of a single IPM does not need to focus on the entire input space of the SUT.

When the tester is satisfied with the parameters and their partitions, it is time to add potential constraints, establish the translation table and add any preselected test cases. These three activities can be performed in any order.

3.5 Document constraints

In order to avoid invalid test cases, any invalid sub-combinations of parameter partitions needs to be documented. Figure 3 shows an example based on the previously described `boolean find_element(list, element)` function. An IPM with two parameters *A*, with four partitions, and *B*, with three partitions, has been identified. Two of the partition combinations do not make sense and are thus invalid. In this example, these are represented as a list of invalid pairs of parameter partitions. In the general case other representations can be used, for example, a set of inequalities.

Params	Partitions			
	1	2	3	4
A: length and contents	one element	more than one unsorted	more than one sorted	more than one all identical
B: match	element not found	element found once	element found more than once	-
Invalid combinations: (A1, B3), (A4, B2)				

Figure 3. Examples of invalid parameter partition combinations

3.6 Establish translation table

The translation table describes the mapping between the partitions of the IPM parameters and input values of the actual test cases.

The translation from abstract to real test cases can be performed either manually or automatically. If manual translation is used, it is often enough to provide a description of the set of actual values that correspond to each IPM parameter partition. When the translation is done, it is up to the tester to select a specific value from the set.

Automatic translation requires more details in the translation table. IPM parameter partitions that contain

more than one value require sampling instructions to be expressed. Further, the translation table needs to be expressed in a machine readable form.

3.7 Add pre-selected test cases

A legacy of test cases for a SUT is not uncommon. For instance, there may be regression test cases. Many of the combination strategies support the use of pre-selected test cases. These combination strategies compute the coverage achieved by the already selected test cases and then add new test cases that increase coverage until the full coverage is reached. The only requirement on a set of preselected test cases is that they are expressed as abstract test cases of the current IPM.

3.8 More IPMs decision

For a complex SUT it can be better to have several small IPMs than one large. This approach allows for a divide-and-conquer strategy when determining the modeling approach and during parameter and partition identification. Another advantage with multiple IPMs for the same SUT is that it allows varying level of coverage.

For instance, one IPM may contain only valid values and another IPM, which is focused on error-handling, may contain invalid values. The valid value IPM may be covered using a higher level of coverage (such as pairwise [8]), and a lower level of coverage (such as one-wise [8]) for the invalid value IPM.

IPMs do not need to be non-overlapping as long as the test cases generated make sense.

4 Initial Input Parameter Modeling Results

Two experiments have been conducted in order to assess the applicability of the proposed process for input parameter modeling.

The aim of the first experiment was to compare the efficiency and effectiveness of several different combination strategies [7]. Five Unix like commands seeded with faults were used as “systems under test”. An early version of the suggested input parameter modeling process was used to create one IPM for each of the systems under test. The IPMs were then input to the different combination strategies, test cases were generated, and finally executed. The faults found were monitored and compared to the known 128 faults. In total 120 of the faults were found by at least one of the combination strategies. Hence, information to detect at least 94% of the faults had been included in the IPMs. Analysis of the eight missed faults provided further insight. Two faults were missed due to automation issues. In the experimental set-up output to stdout and stderr was merged so the error messages appearing on stdout instead of stderr were never detected. Hence, the contents of our IPMs could never affect the detection of these two faults.

At the time of the first experiment, only conflict-free IPMs could be handled. SUTs with conflicts had to be represented by incomplete IPMs. In the experiment, three faults were missed due to this weakness. This result led to a refinement of the input parameter modeling process, i.e., to allow for IPMs to handle conflicts.

The final three faults could have been detected if the IPMs had contained other values, i.e., these faults remained undetected due to the actual modeling. However, these faults only manifested themselves with values that were impossible to derive from the specifications, i.e., these faults are much more white-box related than black-box related.

The two main observations from this experiment is that the input parameter modeling process is feasible to use by experts and that the structured use of an input parameter modeling process is promising with respect to detected faults.

The aim of the second study was to determine how well novices would be able to perform using the input parameter modeling method [6]. Nineteen test professionals, none previously familiar with combination strategies, were given a one-hour tutorial on combination strategies and input parameter modeling. Then an experiment was conducted in which each respondent was asked to construct IPMs from a given specification. The resulting IPMs were then analyzed with respect to correctly applying the concepts, completeness, and overlap. Fifteen of the nineteen subjects managed to create usable IPMs from which test cases could be generated. Three of the four subjects whose IPMs could not be used for test case generation had no prior experience in applying other structured test case selection methods.

Our main observation from this experiment is that with only a small investment in training it is possible to reach a point, for already experienced testers, where input parameter modeling will yield useful results. This is a very promising result since the combination strategy testing process allows the tester to improve the quality of the IPM iteratively. It is also our definite view that the quality of the IPMs will increase as the testers become more accustomed to the input parameter modeling process.

5 Related Work

We know of no input parameter modeling method that is custom-designed to be used with combination strategies. However, several studies have been made of the general problem of identifying parameters and parameter values. Much of this knowledge can apply to combination strategies.

The Category Partition (CP) method [12] has many features in common with our suggested method. Our method differs from CP in providing more guidance in the parameter and parameter value selection. Further, our method supports evaluation of the completeness of the IPM(s).

Two other IPM related methods are Classification Trees [9] and a UML activity diagram based method [3]. Both methods result in the identification of parameters, values and constraints among the parameter values. Neither method supports translation tables or addition of pre-defined test cases.

Beizer [1], Malaiya [10], and Chen et al. [2] also address the problem of parameter and parameter selection but do not describe complete processes.

6 Summary and Conclusions

This work presents an eight-step structured method for the creation of input parameter models custom-designed for combination strategies. This work also reports on initial results from two experiments in which the effectiveness and efficiency of the input parameter modeling method is investigated. The results indicate that the method may lead to capturing enough information in the resulting IPM to reveal most faults in the system under test. The experiment results also indicate that persons with a testing background need only a little tutoring to be able to produce useful, albeit not complete, IPMs based on the suggested process.

7 Future Work

The task of understanding how to perform input parameter modeling has only begun. A number of issues still need further research. One is to further understand the impacts of the different modeling approaches. Another is to continue the work toward the validation of the input parameter modeling method. The results reported in this work is based on two small investigations of qualitative nature. A more quantitative approach would complement the existing results.

Acknowledgements

This research has been founded in part by Enea AB, KK-stiftelsen (The Swedish knowledge foundation) and the University of Skövde. We would also like to express our gratitude to Jonas Mellin of the University of Skövde who provided valuable suggestions and feed-back during the early stages of this work. Offutt is sponsored in part by National Institute of Standards and Technology (NIST), Software Diagnostics and Conformance Testing Division (SDCT).

References

- [1] B. Beizer. *Software Testing Techniques*. Van Nostrand Reinhold, 1990.
- [2] T. Chen, P.-L. Poon, S.-F. Tang, and T. Tse. On the Identification of Categories and Choices for Specification-based Test Case Generation. *Information and Software Technology*, 46(13):887–898, 2004.
- [3] T. Chen, S.-F. Tang, P.-L. Poon, and T. Tse. Identification of Categories and Choices in Activity Diagrams. In *Proceedings of the Fifth International Conference on Quality Software (QSIC 2005) 19-20 September 2005, Melbourne, Australia*, pages 55–63. IEEE Computer Society, September 2005.
- [4] D. M. Cohen, S. R. Dalal, A. Kajla, and G. C. Patton. The automatic efficient test generator (AETG) system. In *Proceedings of Fifth International Symposium on Software Reliability Engineering (ISSRE'94), Los Alamitos, California, USA, November 6-9, 1994*, pages 303–309. IEEE Computer Society, 1994.
- [5] D. M. Cohen, S. R. Dalal, J. Parelius, and G. C. Patton. The Combinatorial Design Approach to Automatic Test Generation. *IEEE Software*, 13(5):83–89, September 1996.
- [6] M. Grindal, Å. G. Dahlstedt, A. J. Offutt, and J. Mellin. Using Combination Strategies for Software Testing in Practice - A proof-of-concept. Technical Report HS-IKI-TR-06-010, School of Humanities and Informatics, University of Skövde, 2006.
- [7] M. Grindal, B. Lindström, A. J. Offutt, and S. F. Andler. An evaluation of combination strategies for test case selection. *Empirical Software Engineering*, Available in SpringerLink electronic Online First version, 2006.
- [8] M. Grindal, A. J. Offutt, and S. F. Andler. Combination testing strategies: A survey. *Software Testing, Verification, and Reliability*, 15(3):167–199, September 2005.
- [9] M. Grochtmann and K. Grimm. Classification Trees for Partition Testing. *Journal of Software Testing, Verification, and Reliability*, 3(2):63–82, 1993.
- [10] Y. K. Malaiya. Antirandom testing: Getting the most out of black-box testing. In *Proceedings of the International Symposium On Software Reliability Engineering, (ISSRE'95), Toulouse, France, Oct, 1995*, pages 86–95, Oct. 1995.
- [11] G. J. Myers. *The Art of Software Testing*. John Wiley and Sons, 1979.
- [12] T. J. Ostrand and M. J. Balcer. The Category-Partition Method for Specifying and Generating Functional Tests. *Communications of the ACM*, 31(6):676–686, June 1988.
- [13] H. Yin, Z. Lebne-Dengel, and Y. K. Malaiya. Automatic Test Generation using Checkpoint Encoding and Antirandom Testing. Technical Report CS-97-116, Colorado State University, 1997.