Thesis Proposal: Evaluation of Combination Strategies for Practical Testing *

Mats Grindal

2004-06-23

Technical Report HS-IKI-TR-04-003

School of Humanities and Informatics
University of Skövde

Abstract

A number of combination strategies have been proposed during the last fifteen years. Combination strategies are test case selection methods where test cases are identified by combining interesting values of the test object's input parameters. Although some results, achieved from small isolated experiments and investigations, point in the direction that these methods are useful in practical testing. Few attempts have been made to investigate these methods under realistic testing conditions. We outline a thesis proposal that is an attempt to determine if combination strategies are feasible alternatives to the currently used test case selection methods in practical testing.

For combination strategies to be feasible alternatives to use in practical testing we require two things. Firstly, the combination strategies need to be effective in finding faults, at least as effective as currently used methods. Secondly, the cost per fault found when using combination strategies should not exceed the corresponding cost for the currently used methods.

To investigate the effectiveness and efficiency of combination strategies we need to establish a benchmark from practical testing and then compare that with how combination strategies perform in the same or similar situations.

Further, we need a testing process targeted for the use of combination strategies to be able to assess the complete cost of using combination strategies. Thus, an important part of this research project is to develop a combination strategies testing process. In particular, the activities

*This research is jointly supported by KK-stiftelsen, Enea Systems AB and the University of Skövde

of selecting combination strategies to use and transforming the requirements on the test object into a format suitable for combination strategies are focused on. These activities are specific to combination strategies and not very well understood.

The methods used for achieving our research goal include literature surveys, investigation of the state-of-practice, with respect to used test case selection methods and cost of testing, experiments, tool implementations, and proof-of-concept, in the form of a case study.

In addition to the direct results of our investigations we expect this research to result in detailed information about how to use the suggested test process. This information will include work instructions covering the manual parts. The process information will also include functional descriptions of the tools as well as interface descriptions of the input and output formats of each tool. These tool descriptions will make the test process generic in the sense that alternative tool implementations can be evaluated keeping everything else constant.

Contents

1 Introduction

1.1 Overview

Combination strategies are test case selection methods where test cases are identified by combining interesting values of the test object input parameters based on some combinatorial strategy. More than ten different combination strategies have been proposed over the last fifteen years, see appendix A. Recently, combination strategies have received an increased attention from the research community [DJK+99, Wil00, LT01, WP01, DHS02, KR02]. The results indicate a wide applicability of combination strategies in testing. However, only a few of these investigations emphasize the practical aspects of using combination strategies in testing. For instance many of the investigations ignore the problem of how to select a few values for each input parameter. Also many of the investigations have been performed using small test problems. Thus, we conclude that many of the practical benefits and limitations of these proposed methods remain to be explored, in particular with respect to practical testing. For instance: Are some combination strategies better than others? If so, are they always better or just in some cases? Are combination strategies simple

enough to use for the average tester? Can enough of the tasks in the process of using combination strategies be automated?

This thesis proposal defines a research project that attempts to determine if combination strategies are feasible alternatives to the currently used test methods in practical testing. The feasibility of using combination strategies in testing is judged by the two factors effectiveness and efficiency. The effectiveness of a test case selection method relates primarily to its ability to select fault revealing test cases. The efficiency of a test case selection method is concerned with its resource consumption. In this thesis proposal, time is considered the primary resource. When comparing combination strategies with other test case selection methods we want to include the time used in all steps of each method. Thus, we need to use the actual time consumption as the unit of comparison. However, in the cases where we want to compare two combination strategies, we can use the number of test cases in the test suites of the two combination strategies as the unit of comparison as an approximation of the time consumption. This is possible since the steps and the contents of each step, when using combination strategies, are the same for both methods.

To be feasible alternatives, combination strategies must provide added value, to the tester, compared to currently used methods. We consider added value to be provided by the use of combination strategies if 1) more faults are found or 2) fewer but previously undetected faults are found or 3) the total cost of using combination strategies is less than the current cost of testing.

The starting point of this research is an assessment of the current way of testing. This is to create a basis for comparison with the use of combination strategies. Further, we need a test process custom-designed for the use of combination strategies in order to realistically evaluate the total cost of testing. The next step of this research project is to develop a combination strategies testing process. Much of the focus of this work will be on activities specific to the use of combination strategies. In particular, how to compare and select an appropriate combination strategy to use and how to prepare the input to the combination strategies, will be focused on. The input to a combination strategy is a representation of some of the requirements of the test object.

Based on the combination strategies testing process a cost model will be developed to make it possible to assess the cost, i.e., the time consumption when using the process. In the scope of this research project, the cost model will be used to compare the performance of combination strategies with the state-of-practice. Alternate uses of the cost-model are fine-tuning of the process and as an aid for estimation of time consumption when planning a test project.

The methods for reaching our objectives include literature surveys, experiments, tool implementations and a final proof of concept in which the complete process is tested and evaluated in a case study.

1.2 Document Outline

Section 2 contains a background on testing in general and the family of test case selection methods called combination strategies in particular. Specifically, the suggested different usages of combi-

nation strategies in testing are described, which suggests that combination strategies are, at least, interesting to consider when faced with a testing problem. Also described, are the different coverage criteria associated with combination strategies. This leads to the problem formulation in section 3, i.e., to determine if combination strategies are feasible alternatives to currently used test case selection methods in practical testing. From our research question a number or research objectives are derived, which are all included in the problem section.

The approaches for reaching our objectives are described in section 4. This section is organized in the same way as the previous problem section to facilitate easy cross-referencing. Following this tradition, the results section (5) is also organized in the same manner. The contents of the results section is a mix of already achieved results and expected results since parts of some questions already have been answered, primarily from surveying previous results in the area. Section 6 points to some related work. Section 7 concludes this research proposal with a summary, highlights of our contributions and some future research directions.

To complete this research proposal there are two appendices. Appendix A contains a classification and short descriptions of the combination strategies for test case selection identified up to date and appendix B contains the collected set of papers reporting on results and experiences from using combination strategies in different settings.

2 Background

A complete description of testing is impossible to give in the scope of this document. However, some key issues, important to this research project are highlighted in the following subsection. It leads to the motivation why combination strategies are interesting. The following subsections then provides a description of what combination strategies really are and how they can be used in different testing scenarios. Some general properties of combination strategies are also explained.

2.1 Testing

Testing is the activity in which test cases are identified, prepared and executed. Identification of test cases is the task of deciding what to test. At least a test case contains some input and an expected result. The most central part of preparing a test case is to determine what to test and document exactly how to execute the test case. The execution of a test case includes following the instructions from the preparation, i.e., feeding the input to the test object, to capture the actual result from the test object and to compare the actual result with the expected result.

Testing consumes a significant amount of the resources in a development project [Mye79, Bei90]. Thus, it is of general interest to assess the effectiveness and efficiency of currently used testing methods and compare these with new or refinements of existing testing methods to find possible ways of improving the testing activity [Mye78, BS87, Rei97, WRBM97, SCSK02]. Traditional metrics for testing effectiveness include number of faults found and achieved coverage, where

coverage is usually related to some property of the test object, e.g., requirements or code. Testing efficiency is related to the time consumption of the whole test activity. However, in theoretical studies focusing on algorithms to identify test cases, i.e., *test case selection methods*, the time consumption is often approximated by the number of test cases generated by the test case selection method.

Several existing test case selection methods (e.g. Equivalence Partitioning [Mye79], Category Partition [OB88], and Domain Testing [Bei90]) are based on the assumption that the input space of the test object may be divided into subsets such that all the points in the same subset result in a similar behavior from the test object. This is called the *partition testing* assumption. Even if the partition test assumption is an idealization it has two important properties. Firstly, it makes it possible, for the tester, to decrease the number of test cases by selecting one or a few test cases from each subset instead of using all possible test cases. Secondly, it gives the tester possibilities to measure the testing effectiveness by using partition coverage. *Partition coverage* is the number of tested partitions divided by the total number of partitions.

An alternative to partition testing is *random testing* in which test cases are chosen randomly based on some input distribution (often uniform distribution) without exploiting any information from the previously chosen test cases nor from the specification. Intuitively, partition testing should be more effective in finding faults than random testing, but Duran and Ntafos [DN84] have shown that, under certain conditions, random testing might be as effective as partition testing. They showed, consistently, only small differences in effectiveness between partition testing methods and random testing. These results were interpreted in favor of random testing since it is generally less work to identify test cases in random testing because partitions do not have to be defined.

However, Hamlet and Taylor [HT90] later investigated the results of Duran and Ntafos and concluded that the their model was unrealistic. One main reason is that the overall failure probability in their model is too high. Hamlet and Taylor thus theoretically strengthened the case for partition testing but made an important point that partition testing can be no better that the information used to define the partitions. Gutjahr [Gut99] followed up on Hamlet's and Taylor's results and showed theoretically that partition testing is consistently more effective than random testing under realistic assumptions.

Lately, results have been produced that favor partition testing over random testing also in the practical case. Reid [Rei97] and Yin, Lebne-Dengel, and Malayia [YLDM97] performed experiments with different partition testing strategies and compared them with random testing. In all cases, random testing is less effective than the investigated partition testing methods.

Key issues in any partition testing approach are how partitions should be identified and how samples should be selected from the partitions. In early partition test case selection methods, like Equivalence Partitioning (EP) [Mye79] and Boundary Value Analysis (BVA) [Mye79], specifications are used to identify the parameters of the test problem. The identified parameters are then analyzed one by one to determine suitable partitions of each parameter. A weakness with these methods is that the support for identifying parameters and their partitions from arbitrary specifications is rather limited. To handle this, Ostrand and Balcer proposed the Category Par-

tition method (CP) [OB88]. CP consists of a number of manual steps in which an equivalence class model for the parameters of a test object is systematically derived from a natural language specification. In addition to the identified equivalence classes of each parameter, the equivalence class model may also be augmented with selector expressions. The selector expressions may contain information about certain sub-combinations that must be avoided in the final test suite. The selector expressions are used by the test case generator to form complete test inputs by combining partitions of the different parameters of the model such that the constraints defined by the selector expressions are not violated. One of the main shortcomings with CP is that every combination that is valid according to the selector expressions is included in the final test suite, which leads to a combinatorial explosion when the number of parameters and partitions increase. Choosing all [valid] combinations of interesting values is usually impractical. Consider an example where we have five parameters, each with ten interesting values. To try all combinations we would need $10 \times 10 \times 10 \times 10 \times 10 = 100.000$ tests, which in most practical test projects is far too much. To remedy this problem a number of different combination strategies have been proposed. Combination strategies will be thoroughly described in the next section.

2.2 Combination Strategies

As was described in section 1.1, the class of test case selection methods where test cases are identified by combining interesting values of the test object input parameters based on some combinatorial strategy is called combination strategies. Another view of combination strategies is that they are ways to sample the complete set of combinations of values of the parameters of the test problem. Combination strategies include, but are not limited to, techniques from experimental design [Tag87] to choose test cases. The application of orthogonal arrays in testing [Man85] is one example. Orthogonal arrays are tables filled with numbers according to certain rules. Each position in the table is then used to represent a test case by converting the index and the contents of the position into parameters and interesting values of the parameters. A more thorough description of orthogonal arrays can be found in appendix A.2.

The main function of combination strategies is to bring the number of used combinations down to a feasible level. As such, combination strategies have been proposed for usage in several different testing settings. Kropp, Koopman, and Siewiorek [KKS98] as well as Brownlie, Prowse, and Phadke [BPP92] show that combination strategies may be used for robustness testing. The *robustness* of a system is the degree to which is functions correctly in the presence of exceptional inputs or stressful behavior. Williams and Probert [WP96] illustrate how combination strategies may be of use in configuration testing. In *configuration testing*, the same set of test cases are executed on several different software or hardware configurations of the test object. Here, combination strategies are used to identify the configurations that should be tested.

Daley, Hoffman, and Strooper use combination strategies to test Java classes [DHS02]. Combination strategies have also been suggested as a way to select test cases in functional testing, for instance by Ammann and Offutt [AO94], Burroughs, Jain, and Ericson [BJE94], and Cohen,

Dalal, Fredman, and Patton [CDFP97]. Dalal, Jain, Karunanithi, Leaton, Lott, Patton, and Horowitz [DJK+99] sketch how combination strategies may be applied in unit testing. The main difference between unit testing and functional testing from a combination strategy perspective is that in unit testing the actual parameters of the software component under test is the starting point for the creation of the test cases, whereas in functional testing the starting point is the functional specification of the test object. Dalal et al. also give some details on how combination strategies may be applied when testing is based on operational profiles consisting of a number of steps.

2.3 Coverage Criteria of Combination Strategies

Like many test case selection methods, combination strategies are based on coverage. In the case of combination strategies, coverage is determined with respect to the values of the parameters of the test object that the tester decides are interesting.

The simplest coverage criterion, i.e., each-used coverage, does not take into account how interesting values of different parameters are combined, while the more complex coverage criteria, such as pair-wise coverage, is concerned with (sub-)combinations of interesting values of different parameters. The following subsections define the coverage criteria satisfied by combination strategies included in this paper.

Each-used (also known as 1-wise) coverage is the simplest coverage criterion. 100% each-used coverage requires that every interesting value of every parameter is included in at least one test case in the test suite.

100% *Pair-wise* (also known as 2-wise) coverage requires that every possible pair of interesting values of any two parameters are included in some test case. Note that the same test case may cover more than one unique pair of values.

A natural extension of pair-wise (2-wise) coverage is *t-wise* coverage, which requires every possible combination of interesting values of t parameters be included in some test case in the test suite. t-wise coverage is formally defined by Williams and Probert [WP01].

A special case of t-wise coverage is *N-wise* coverage, where N is the number of parameters of the test object. *N-wise coverage* requires all possible combinations of all interesting values of the N parameters be included in the test suite.

The each-used, pair-wise, t-wise, and N-wise coverage criteria are purely combinatorial and do not use any semantic information. More coverage criteria can be defined by using semantic information. Cohen et al. [CDFP97] indicate that valid and error parameter values should be treated differently with respect to coverage. *Normal* values lie within the bounds of normal operation of the test object, and *error* values lie outside of the normal operating range. Often, an error value will result in some kind of error message and the termination of the execution. To avoid one error value masking another Cohen et al. suggest that only one error value of any parameter should be included in each test case. This observation was also made and explained in an experiment by Grindal et al. [GLOA03].

By considering only the valid values, a family of coverage criteria corresponding to the general t-wise coverage criteria can be obtained. For instance, 100% *each valid used* coverage requires every valid value of every parameter to be included in at least one test case in which the rest of the values are also valid. Correspondingly, 100% *t-wise valid* coverage requires every possible combination of valid values of t parameters to be included in some test case, and the rest of the values are valid.

Error values may also be considered when defining coverage criteria. A test suite satisfies *single error* coverage if each error value of every parameter is included in some test case in which the rest of the values are valid.

Ammann and Offutt used a special case of normal values to define base choice coverage. First, choosing the most frequently used value of each parameter identifies a base test case. We assume here that the most frequently used value of each parameter is a normal value. 100% *base choice* coverage requires every interesting value of each parameter to be included in a test case in which the rest of the values are base values. Further the test suite must also contain the base test case.

3 Problem

In appendix A a number of investigations of combination strategies are collected. Among these investigations there are a few that report on the applicability of combination strategies in practical testing. Usually these papers describe how a specific combination strategy has been used in a practical problem. Despite, being some kind of proof-of-concept reports these reports do not address the question of how to use combination strategies in the general case. Neither do these reports give much insight into the effectiveness and efficiency of combination strategies compared to other test case selection methods. Most reports lack comparisons with other test case methods [KKS98, WP96, DHS02, AO94, DJK+99]. Only a few reports contain any comparative studies [BPP92, BJE94, CDFP97]. However, Brownlie, Prowse, and Phadke [BPP92] base their comparisons on estimated results of a traditional approach. Whereas Cohen, Dalal, Fredman, and Patton [CDFP97], although comparing the results of employing combination strategies with results from real traditional testing, do not describe the traditional testing. Finally, Burroughs, Jain, and Erickson [BJE94], describe two "traditional" test case selection methods and compare the number of test cases generated by these with the number of test cases generated by a combination strategy for two small examples. No comparison of the effectiveness of the generated test cases is performed.

Some attempts have been made to assess the efficiency and effectiveness of combination strategies compared to other methods. Dunietz, Ehrlich, Szablak, Mallows, and Iannino [DES+97], Kuhn and Reilly [KR02], and Grindal, Lindström, Offutt, and Andler [GLOA03] all show results that indicate that under certain conditions in practice test suites from combination strategies may be much more efficient and nearly as effective as test suites containing all possible combinations. For instance Grindal et al. investigated four combination strategies that detected 108 to 119 of

the 120 known faults with test suites ranging in size between 30 and 181 test cases of 6480 possible test cases. Although more focused on comparing combination strategies with each other and with other test case selection methods, in particular "all combinations" and random testing these reports lack a holistic perspective, i.e., the cost and results of the complete test process employed to real sized test problems. For instance, the problem of identifying parameters and representative values of each parameter is almost ignored in these reports. Also the questions of which tasks to automate and how to do it are overlooked in these studies.

Despite these fragments of knowledge of combination strategies and their applicability we draw the initial conclusion that it is not known if combination strategies are feasible alternatives to other test methods in practical testing. The aim of this thesis proposal is to find out the answer to the question:

Are combination strategies feasible alternatives to other test methods in practical testing?
To be feasible alternatives we claim that the following criteria must be satisfied:

1 How to use combination strategies must be described - a combination strategies testing process.

2 The combination strategies testing process must be complete with respect to its contained activities

3 Using the combination strategy testing process must provide added value, to the tester, compared to currently used methods

The following sections will provide more details on these three criteria, which will lead to the formulation of a number of research objectives.

3.1 A Combination Strategies Testing Process

When defining a general process for using combination strategies we must start by looking at how testing is performed in practice. If a general process for using combination strategies does not fit within the "standard" way of testing, it is likely that the defined process will remain unused. Figure 1 shows a simple test process definition that is generic enough to include how most testing is performed in practice.

The first step of any testing process is to plan the forthcoming activities. The planning includes, at least, identifying the tasks to be performed, estimating the amount of resources needed to perform the tasks, and making economic and time budgets. The second step of the test process is to make any preparations needed for the upcoming test execution. The main tasks during the preparation step are to select and document the test cases. In the third step, the test cases are executed and results are collected. These results are then analyzed in the fourth, and last, step. There are at least two levels of analysis. In the low level the results of a single test case may be analyzed to determine if a problem exist and should be reported. In the high level the results from many test cases are analyzed in order to determine if testing can be terminated. Since both low

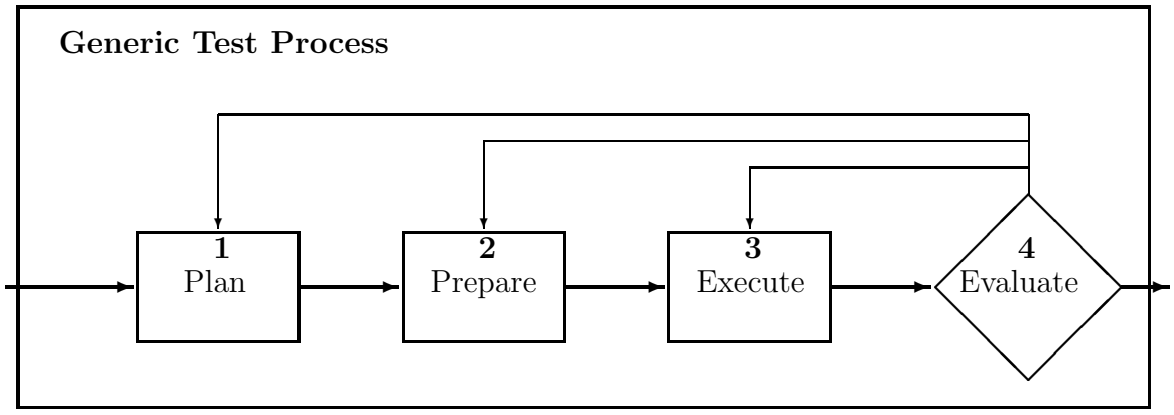

Figure 1: A Generic Testing Process

level and high level analysis may result in the need for more testing there are several feed-back loops in the process description. A discovered fault may result in re-execution of the same test case after debugging. Too little testing may require more test cases and possibly replanning. This simple test process leads to the formulation of an initial research objective:

Our first research objective is to define a combination strategies testing process.

The definition of the combination strategies testing process includes a listing of the tasks of the process, information about the order of the tasks and for each task a description of that task.

A main requirement on a combination strategies testing process is that it conforms to the generic test process described in figure 1. However, although the knowledge that combination strategies should be used may have an impact on the planning tasks, for the completion of this research we will not require the combination strategies testing process to include a specific planning task. The main reason is that planning may be performed in a large variety of ways and the combination strategies testing process should not impose any unnecessary restrictions on the planning.

A secondary requirement on a combination strategies testing process is that all tasks of the process are described in such detail that each task, when performed, yield well-defined results. In this context, a well-defined result is a result that enables the next task to be performed. In the scope of this research project, not all tasks of the defined combination strategies testing process need to be described in full detail. Describing tasks or subtasks with the only purpose of increasing the efficiency of the process as a whole, for example automatic execution, is considered optional.

Two activities that are specific for combination strategies and thus must be included in a combination strategies testing process are to select which combination strategy to use and to transform [parts of] the specification into a format suitable for combination strategies. These two activities will be described in greater depth in the following sections.

3.2 Combination Strategy Selection

Selecting which combination strategy to use, faced with a testing problem is not trivial. Appendix A contains more than ten different combination strategies all with different performance. Further, Cohen et al. [CDFP97] as well as Grindal et al. [GLOA03] show the benefits of using more than one combination strategy, increasing the alternatives for the tester.

The used combination strategies will greatly impact the effectiveness and efficiency of the whole test activity. Found faults and achieved coverage are two often used testing effectiveness metrics. Grindal et al. [GLOA03] show for some combination strategies that different combination strategies may target different types of faults. Different combination strategies also have different coverage criteria associated. This is shown in section 2.3. Thus, both targeted faults and type of coverage supported by combination strategies are important to consider when selecting which combination strategies to use.

A finite amount of time allocated for the testing will set a practical limit to the number of test cases that is possible to prepare and execute. Thus, the size of the generated test suite is an important factor to consider when selecting the combination strategies to use since different combination strategies will generate test suites of different sizes [GLOA03].

The algorithms of some of the more complex combination strategies, e.g., IPO [LT98] and AETG [CDPP96] are quite time consuming. It may be the case that complexity of the algorithms may affect the efficiency of the testing. Thus, the algorithm complexity and its relation to the total time consumption of the test process also need to be investigated.

Some test problems may have restrictions in how parameters values may be combined. Thus, the combination strategies need to support parameter value conflict handling. However, the underlying algorithms of the combination strategies work differently, some, like OA [Man85], generates the whole test suite at once, others, like AETG [CDPP96], generate one test case at a time, and yet others, like IPO [LT98], builds the test suite by covering one parameter at a time. The different algorithms of the combination strategies lead to different approaches in conflict handling. A first question is if the associated conflict handling with a given combination strategy is expressive enough to handle the test problem. If the expressiveness of the conflict handling is sufficient, how is the number of test cases in the final test suite affected by the conflict handling mechanism.

There is no doubt that the specific situation in which a test problem is to be solved will affect which properties of the combination strategies that are important to evaluate. In some test projects time is the most important resource, which may lead to the choice of a highly automated combination strategy or a combination strategy that generates few test cases. In other projects, the quality of the product is prioritized which may lead to the choice of a combination strategy which detects many faults of a certain kind or a combination strategy which yields high parameter coverage. It is difficult to devise a general policy for combination strategy selection. However, by assessing a number of important properties it is possible to create a basis for the comparison and selection of combination strategies to solve a specific test problem.

Thus, we draw the conclusion that both the testing effectiveness and the testing efficiency is

greatly affected by the choice of combination strategies. To be able to decide on which combination strategy to use for a specific test problem the tester needs to be able to compare different combination strategies on the grounds of testing effectiveness and efficiency. This leads to the formulation of another research objective.

Our second objective of this research is to create possibilities for assessing a number of important properties of combination strategies.

At least the following properties should be investigated:

- Targeted faults
- Supported parameter coverage criteria
- Size of generated test suite
- Time complexity of the test selection algorithm
- Support for and performance of the associated parameter value conflict handling methods

The primary goal when investigating these properties is to make quantitative assessments, i.e., some forms of metrics of the chosen set of properties of the combination strategies. If that is not possible for all properties our secondary goal is to find a [partial] ordering between the different combination strategies with respect to each such property. For properties where such an ordering is not possible in general, for instance due to different results for different test objects, an evaluation method for each such property should be proposed and demonstrated.

3.3 Input Parameter Modeling

The use of combination strategies requires the test problem to be represented as a number of dimensions, each with a finite number of values. The reason is that the algorithms of the combination strategies are based on selecting points from an n-dimensional finite space. Thus, the tester faces the task of mapping the test problem onto the axes of a co-ordinate system with n dimensions. In the general method to do this, parameters of the test problem are identified and represented as different dimensions in the co-ordinate system. Representative values of each identified test problem parameter are selected and enumerated to map them onto the values of the corresponding dimension.

At a first glance, identifying the parameters of a test problem seems like an easy task. Almost all software components have some input parameters, which could be used directly. This is also the case used in most of the works on combination strategies [KKS98].

However, Yin, Lebne-Dengel, and Malayia [YLDM97] point out that in choosing a set of parameters the problem space should be divided in sub-domains that conceptually can be seen as consisting of orthogonal dimensions. These dimensions do not necessarily map one-to-one onto the actual input parameters of the implementation. For instance, Cohen, Dalal, Parelius, and Patton [CDPP96] state that in choosing the parameters, one should model the system's functionality not its interface.

To illustrate the difference, consider the case in which some of the functionality of an ATM is to be tested. One of the input parameters of this test problem is the amount of money to be withdrawn. Another input parameter of this test problem is the amount of money on the account. Thus, the direct approach for the tester is to use these two parameters and map them onto different dimensions in the co-ordinate system. However, the functionality of the ATM is affected by a combination of the values since withdrawal is allowed only if the account contains enough money. This illustrates a possibility for the tester, i.e., to use an abstract parameter. This abstract parameter would represent the outcome of the ATM transaction, i.e., withdrawal granted or denied. Dunietz, Ehrlich, Szablak, Mallows, and Iannino [DES+97] show that the same test problem may result in several different parameter-value representations depending on which [abstract] parameters that are used to describe the test problem.

Finding some representative values for a certain parameter may also be done in a number of ways. The papers by DeMillo, Lipton, and Sayward [DLS78] and Myers [Mye78] provide useful hints on how to choose values. Further Equivalence Partitioning [Mye79] and Boundary Value Analysis [Mye79] are two methods that can be used. The activity of finding a suitable set of parameters and parameter values is called *input parameter modeling*.

Daley, Hoffman, and Strooper [DHS02] claim that creating the input parameter model is the most important step in test preparation. The reason for their claim is that the resulting input parameter model will have an impact on both the testing effectiveness and the testing efficiency. For instance, the number of parameters and the number of values identified for each parameter will impact the amount of test cases in the test suite created by a combination strategy thus affecting the time consumption of the testing. Further, the chosen parameter values may affect which faults that are detected by a test suite [GLOA03].

The task of creating an input parameter model may be further complicated by constraints in the test case space. Cohen, Dalal, Fredman, and Patton [CDFP97] show examples in which a specific value of one of the identified parameters is in conflict with one or more values of another parameter. In other words some (sub-) combinations of parameter values may not be feasible. Hence, the input parameter model must support constraints to be expressed.

As illustrated by the above examples the tester is faced with a number of decisions when designing an input parameter model. Some of these decisions will impact the efficiency and effectiveness of the testing. Thus, the tester needs a way to predict the consequences of these decisions. This leads to the formulation of a third research objective.

Our third objective in this research is to define a structured method for making an input parameter model of a test problem.

A number of requirements on the structured method can be formulated: (1) The result from the input parameter modeling method should allow any combination strategy to be used. Some of the combination strategies, for instance the base choice strategy, require some semantic information to be explicitly expressed, which adds to information that needs to be included in the input parameter model if any combination strategy should be possible to use. (2) A large number of specifications are expressed in natural language. Thus it must be possible to use a natural language specification

as input to the structured method. (3) It must be possible to express relations between parameter values to exclude certain parameter value combinations. (4) The method should convey the impact on testing effectiveness and testing effectiveness whenever there is a decision to be made by the tester. (5) In some cases, the tester has some favorite test cases that (s)he wants included in the final test suite. The input parameter model should include facilities to express such needs.

3.4 Efficiency and Effectivity of Combination Strategies

A tester will not start to use combination strategies unless there is a good chance of increasing the quality of the testing, i.e., combination strategies must provide added value, to the tester, compared to currently used methods. We consider added value to be provided by the use of combination strategies if 1) more faults are found or 2) previously undetected faults are found or 3) the total cost-effectiveness of combination strategies are better than that of the currently used methods. Thus we need to focus both on properties of combination strategies and the efficiency of the whole combination strategies testing process. To be able to determine if the use of combination strategies is better than the currently used test selection methods we need to know what is used today and what the use of combination strategies may offer. This leads to the formulation of our two last research objectives.

Our fourth research objective is to assess the state-of-practice with respect to used testing methods and the current cost of testing.

Our fifth objective of this research is to create a cost model for the use of the combination strategies testing process.

The cost model for the use of combination strategies testing process should include estimates of the cost for each of the steps in the process. An important factor for the efficiency of a testing activity is automation. Automation usually requires information to be expressed in some structured way with defined semantics. Thus, on one hand, the initial work to prepare for automation is probably more expensive than the corresponding manual work. On the other hand, when all the preparations have been completed, the actual work is less expensive to perform automatically than manually. Consider for example the task of executing test cases. To prepare a test case for automatic execution a test script need to be written. It is likely that it will take more time to write the test script than to identify the steps that need to be performed during manual execution. However, once the test script is written the automatic execution of the test script is likely to be faster than the corresponding manual execution. Thus, for each step in the process, we need to consider the impact of automation on testing efficiency, and it should be possible to see the effects of automation of a task in the cost model.

3.5 Summary of Research Objectives

The objectives of this research, described in the previous sections, are summarized in table 1.

No.	Description
1	To define a combination strategies testing process
2	To create possibilities for assessing a number of important properties of combination strategies
3	To define a structured method for making an input parameter model of a test problem
4	To assess the state-of-practice with respect to used testing methods and the current cost of testing
5	To create a cost model for the use of the combination strategies testing process

Table 1: Summary of objectives of this research project.

4 Approach

To recapitulate our initial research problem we intend to find out if combination strategies are feasible alternatives to other test methods in practical testing. By identifying the five research objectives, summarized in section 3.5 we have initiated a divide-and-conquer approach to finding an answer to our research question. The following sections will describe the methods we intend to use in order to reach these objectives.

4.1 A Combination Strategies Testing Process

The work of creating a combination strategies testing process starts in the existing reports on applications of combination strategies. The first part of this work is to identify the tasks of the process and the order in which the tasks should be performed. As was stated in section 3.1 the suggested process should conform to the generic test process presented in figure 1. Further, the process must tasks to accomplish the activities of selecting combination strategies (see section 3.2) and creating an input parameter model (see section 3.3). In section 5.1 an initial proposition of a combination strategies testing process fulfilling these requirements is described.

The next step in this part of the research is to detail and describe each step of the process. The main method for achieving this is to study existing solutions to each of the tasks to find suitable ways of performing each task. The tasks of combination strategy selection and input parameter modeling, being central to this research project, will get further attention in the forthcoming sections.

The final part of the work with creating a combination strategies testing process is to validate it. Using the testing process in a real testing project will be our proof-of-concept. Further details on the proof-of-concept can be found in section 4.4.3.

4.2 Combination Strategy Selection

As outlined in section 3.2, the tester needs ways of comparing properties of different combination strategies in order to make an informed decision of which combination strategy to use. Thus, we need to know which properties that are important to compare. Section 3.2 lists five different properties identified from the published experiences and evaluations in appendix B. It is our assumption that these are the important properties that need evaluation methods. However, to check that validity of our assumption, we intend to ask test practitioners about their test strategies and how test methods are being selected for usage. Should this investigation result in other properties being important for the selection of test methods, then these properties should be added to the list of properties that should be handled within the scope of this work.

Our primary goal, as mentioned in section 3.2, is to make quantitative assessments of the these properties. Thus, we will attempt to identify methods that can be used to measure the combination strategies with respect to these properties. Each property, for which an assessment method is found, all of the combination strategies in appendix A will be evaluated and the results will be included in the description of the task to select combination strategies.

For the remaining properties [partial] orderings of the combination strategies will be attempted. For properties of combination strategies that vary with the actual test problem, for instance achieved code coverage, the order of two combination strategies with respect to that property may vary depending on the used test object. Thus, it may not be possible to calculate a general value for that property nor to establish a general ordering. The best we can do then is to devise and demonstrate an evaluation method in which some properties of the test object are used to assess the candidate combination strategies for that specific test problem.

In the following subsections specific details of the work with different properties are outlined. The ordering of the properties follows the ordering in section 3.2 but has no other significance.

4.2.1 Targeted Faults

A couple of fault taxonomies, or embryos to such, with respect to combination strategies have been suggested. Kuhn et al. [KR02] and Grindal et al. [GLOA03] describe how faults can be classified according to the number of parameters involved in triggering the failure. Kropp et al. [KKS98] focus on faults that result in crashes and non-terminations.

We intend to investigate the different combination strategies with respect to the types of fault they discover. In a first attempt will use the number of parameter taxonomy to classify the combination strategies with respect to detected faults.

Kuhn et al. [KR02] based their experiment on existing fault reports from two software projects. As an optional extension to this research project we will use their approach to see if there are other patterns, apart from number of parameters involved in triggering failures, among the fault reports of some real projects. If patterns exist this may lead to the formulation of another fault taxonomy, which can be used to classify combination strategies.

4.2.2 Parameter Coverage Criteria

The work of identifying satisfied coverage criteria for each combination strategy is a theoretical task, which builds on the seminal work of each of the investigated combination strategies. Most proposed combination strategies already have associated coverage criteria. An overview of the existing coverage criteria can be found in section 2.3. The remaining work to be done in the scope of this research is identifying the coverage criteria of new combination strategies that might be proposed and finding a way of presenting this information in a comprehensive manner.

4.2.3 Size of Generated Test Suite

The different combination strategy algorithms will be investigated in order to assess the anticipated number of test cases generated as a function of the number of parameters and the number of values of each parameter. Some of these results have already been collected, see section 5.2.2. Thus, the focus in this work will be on investigating the remaining combination strategies to find approximate values or at least some min and max number of test cases. In this work we will at least go up to t-wise coverage, where $t = 6$, based on the results of Kuhn and Reilly [KR02].

4.2.4 Time Complexity of Combination Strategy Algorithms

Lei and Tai [LT98] claim that an advantage with IPO over AETG [CDPP96] is the lower time complexity of the test case generation algorithm. In the scope of this research we will investigate the actual time consumption of these algorithms to see if this is a problem in practice. If this is the case we will extend our timing analysis of the test case generation algorithms to include the remaining combination strategy algorithms. We will also include the results in the basis for comparison of combination strategies.

4.2.5 Parameter Value Conflict Handling

As was suggested in section 3.2 sometimes there are subcombinations of parameter values that are infeasible or in some other way not desirable. Thus, there is a need for handling such conflicts. In the scope of combination strategies there are five methods of parameter value conflict handling described.

- **Avoid** selecting a test case invalidated by a constraint [CDFP97].
- **Replace** an invalid combination with others, preserving the satisfied coverage.
- **Change** one or more of the values involved in an invalid combination [AO94].
- **Rewrite** the test problem into several conflict free sub problems [CDFP97, DHS02, WP96].
- **Arrange** the parameters in a hierarchical manner called *classification trees* [GG93] to avoid conflicts.

In section 4.3 the work on the input parameter model is described. One suggested part of the model is the constraint model in which the constraints on combining certain parameter valued are expressed. Using the constraint model, our approach is to investigate if the five constraint handling methods are general enough to be used with any combination strategy. Further, the performance in terms of number of test cases in the final test suites will investigated in general terms or in an experiment, in which several different test problems with different types of constraints will be used.

4.3 Input Parameter Modeling

An essential activity when using combination strategies is to create an input parameter model. The input parameter model contains all information needed by the combination strategies to generate the test cases. Since generation of test cases is automated, the information of the input parameter model needs to be structured according to some rules. Thus our research includes both finding out which information is needed by the combination strategies and devise a suitable structure for this information.

The first step of this work is to find out which semantic information is needed by the different combination strategies. The next step is to identify the different types of relations among the different parameter values that are needed by the combination strategies. Both these steps will be performed through theoretical analysis of the described and suggested combination strategies listed in appendix A. Existing tools for generation of combination strategy test cases is another important source of information for this task, e.g., the research implementations T-GEN [TKG+90], PairTest [LT01], and OATS [BPP92], and the commercial implementation $AETG^{SM}Web$ [1].

The Category Partition method (CP) [OB88] is the only method, known to the author, that transforms a specification expressed in natural language into an input parameter model in a structured manner. CP was originally created to solve testing problems in a manner very similar to combination strategies. Thus, CP is a good start when finding a structured method for identifying the parameters and the parameter values of test problem.

When the requirements on the method for identifying parameters and parameter values have been established the next step is to examine CP to see if it satisfies all of the requirements. If that is not the case, we will attempt to perform some customization of CP to ensure that all requirements are satisfied.

Next step in the work of creating an input parameter modeling method is the creation of a language sufficient for describing the complete input parameter model. A suggested structure for the input parameter model can be found in section 5.3

The work with creating an input parameter modeling method will be concluded with an experiment in which several persons will be given the same specification and asked to follow the method. The results will then be compared and analyzed in order to find out if the different results are similar enough for us to conclude that our method is reasonably deterministic.

[1] http://aetg2.agreenhouse.com, page visited July 2003

4.4 Efficiency and Effectivity of Combination Strategies

As stated previously, our ultimate goal with this research is to decide if combination strategies are feasible alternatives to currently used test selection methods. This requires us to compare the efficiency and effectivity of the state-of-practice with the corresponding performance of combination strategies. Thus, we need to assess the state-of-practice. Further we need a means of determining the performance of combination strategies. The following subsections will deal with these two issues.

4.4.1 State-of-Practice

The concept of state-of-practice is elusive since there is probably as many ways to test, as there are test projects. Thus, there is a problem deciding the representativity of the results no matter the thoroughness of the investigation. For practical reasons we will limit ourselves to investigate companies with testing departments residing in Stockholm, using the customer stock of the consultancy company Enea Systems as primary source of subjects. We are aware that this constraint may limit the value of the results. To reduce the impact of the limitations in selecting subjects we will attempt to achieve a dispersed set of subjects. Examples of properties we will use to enforce disparity are, age and size of organization, age and size of product, and type of product. Another way of reducing the impact of a narrow subject selection is to display the original results in the form of ranges, rather than trying to compute means or medians.

The investigation of the state-of-practice in testing will be conducted as interviews with testers/test project managers at the selected companies.

4.4.2 Cost Model for the Combination Strategies Testing Process

Since this research project does not allow full-scale experiments with different combination strategies our intention is to develop a cost-model for the final combination strategies testing process. Our intention is to use time as our way of determining cost.

For each task of the process, the main contributors to the time consumption of the task will be identified. Using these, as parameters, formulas for time consumption will be constructed for the tasks. Validation of the formulas and their corresponding results will be performed by reviews carried out by experienced testers and comparisons with the time consumption of similar tasks in real testing projects.

For several of the tasks, automation will be an important factor influencing the time consumption, obvious examples are generation of test case inputs, generation of expected results, and test case execution. Thus, the cost model must include the effect of automation of one or several tasks. However, automation of one task may influence another task. Consider automation of the execution. Obviously the task of executing the test cases is speeded-up, but at the cost of creating test scripts, which is usually performed during the preparations step. Thus, the formulas describing the cost of each task in the process cannot be independent.

4.4.3 Proof-of-Concept

A number of research objectives were defined in section 3 that have to be met in order for us to consider our aim reached. However, reaching these objectives individually will not necessarily mean that our aim has been reached. So to strengthen our case, we will perform a proof-of-concept in which the results of this research project, i.e., the test process, complete with work instructions and tools, will be applied in a real testing project. The results from this case study will be evaluated. In particular, we will focus on the effectiveness, i.e., fault-finding ability of the used combination strategies, and the validity of the cost-model. Our hypothesis is that these results will favor the use of combination strategies in practical testing.

Ideally, we would like to execute our case study in a real industrial project in parallel with the organization's normal way of testing so that the results can be compared. However, due to the costs of staging such an experiment we may need to settle with a situation in which our process based on combination strategies is used instead of the normal test method of the organization. To achieve realism and independence, the subject of the case study will be chosen from the customer stock of Enea Systems AB.

The case study will be primarily evaluated on the time consumption of the whole test process and the amount of faults found. If no parallel testing exists, we will compare the results from the case study with results from previous projects executed in the same organization.

4.5 Summary of Activities and Time Plan

Table 2 shows a summary of the activities planned within this research project. To help the reader to make the connection between the activities and the research objectives, the titles of the previously described sections are used to group the activities. Each activity is also labeled with a letter and a number to make the time plan in figure 2 more comprehensive.

5 Expected Results

In some areas of the proposed research there already exists some preliminary results. For instance, the tasks of a suggested combination strategies testing process have been identified. Also the seminal work on the different combination strategies (see appendix B) sometimes contain information on test suite sizes, satisfied coverage, and algorithm complexity. Therefore this section contains a mix of already achieved results and expected results.

5.1 A Combination Strategies Testing Process

One of the main deliverables from this research project is the description of how combination strategies should be used in practice - a combination strategies testing process. Figure 3 shows the tasks and their interdependencies in a suggested combination strategies testing process.

Label	Description	Comment
A.	**Combination Strategies Testing Process**	**-**
A1.	Identify tasks of process	DONE
A2.	Detail the tasks of process	
A3.	Validation of the process	included in D3
B.	**Combination Strategy Selection**	**-**
B1.	Are other properties important	included in D1
B2.	Compare targeted faults	
B3.	Compare parameter coverage criteria	PARTLY DONE
B4.	Compare size of generated test suite	PARTLY DONE
B5.	Compare time complexity of the test selection algorithm	
B6.	Compare support for and performance of the associated parameter value conflict handling methods	
C.	**Input Parameter Modeling**	**-**
C1.	Identify types of information that is needed in the model	
C2.	Identify relations that need to be expressed in the model	
C3.	Define model, syntax, semantics	
C4.	Propose method of transforming specification into model	
C5.	Validate method	included in D3
D.	**Efficiency and Effectivity of Combination Strategies**	**-**
D1.	Investigate state-of-practice	
D2.	Create a cost model	
D3.	Proof-of-concept	

Table 2: Summary of objectives of this research project.

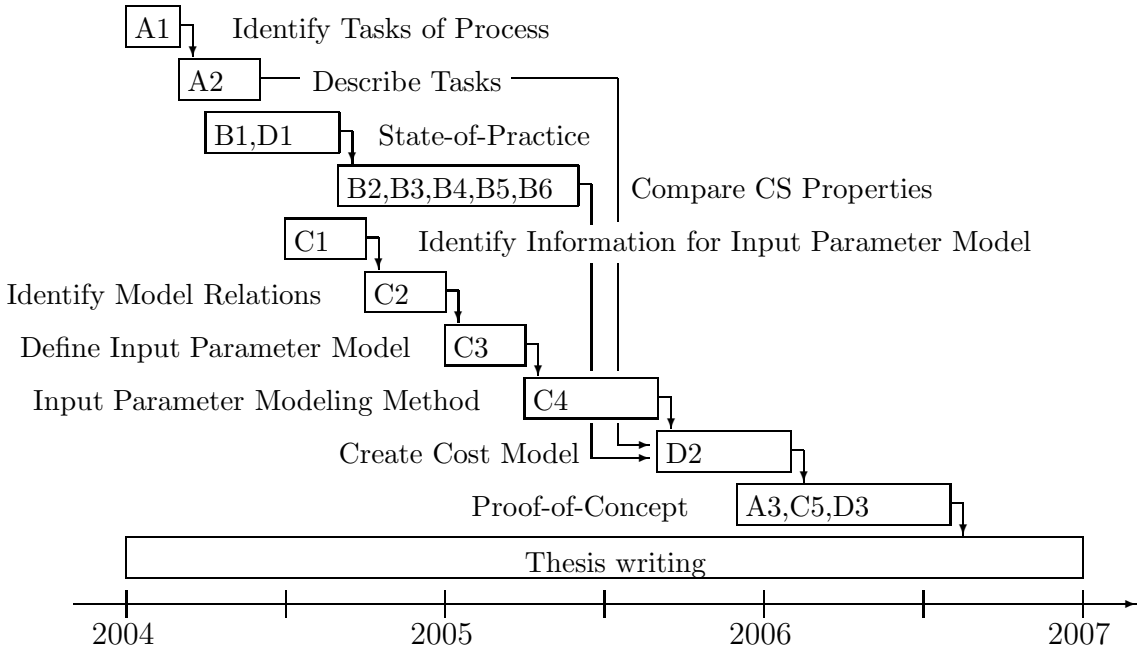

Figure 2: A preliminary time plan for the tasks, and their relations, defined in this research proposal. Keys reference activities in table 2

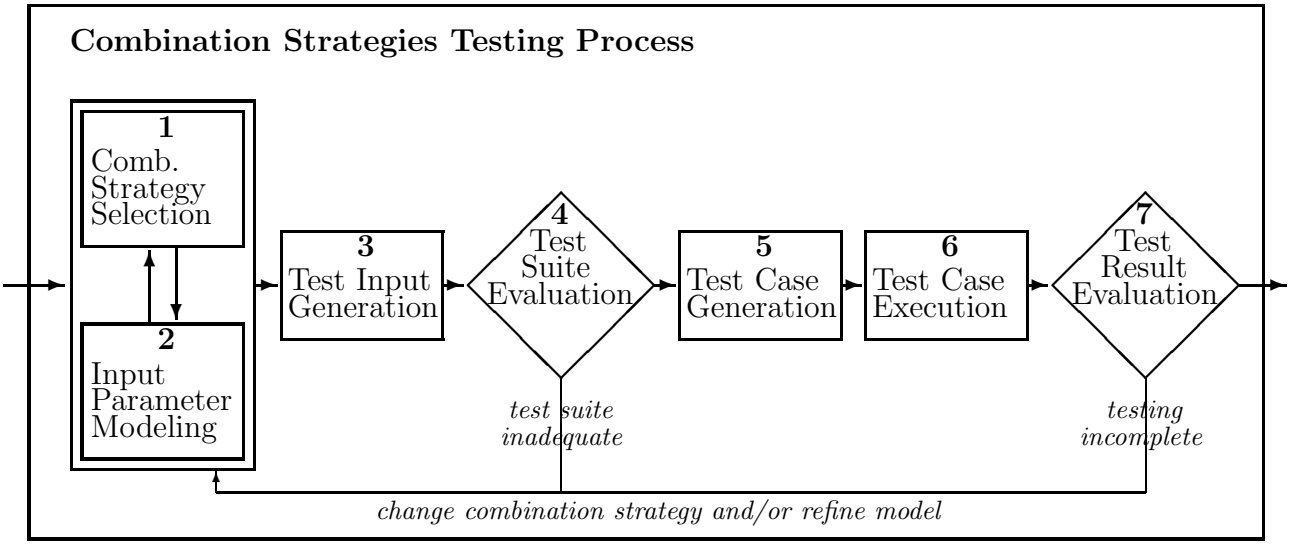

Figure 3: A Process for Applying Combination Strategies in Practical Testing

In step one of the test process one or more combination strategies are chosen to be used in the testing. In step two of the test process, the test problem is analyzed and an input parameter model is constructed. The input parameter model describes the different parameters of the test problem. For each parameter, a number of values are selected to represent different aspects of that parameter, for instance valid and invalid inputs. Input parameter modeling also includes identifying constraints among the values of the different parameters. In step three of the test process a number of test inputs are generated by applying the chosen combination strategies to the input parameter model. The task of generating test inputs is fairly cheap to automate, see for instance the experiment conducted by Grindal et al. [GLOA03].

The relative ease of generating test inputs, i.e., an incomplete test suite, is exploited in step four of the combination strategies testing process. In this step the incomplete test suite may be evaluated with respect to a number of different static properties, for instance actual size of test suite, coverage with respect to input parameter values etc. It is also possible for the tester to execute the inputs, thus investigating dynamic properties of the incomplete test suite, e.g., code coverage. Note, however, that executing the test suite at this stage will not yield any pass/fail results since expected results are not yet defined for the test cases. Should the evaluation of the incomplete test suite be unsatisfactory with respect to some chosen criterion the tester may return to the first two steps of the process. Thus the tester can fine-tune the test suite before attempting to complete each test case, by adding expected results, in step five of the process. We expect this possibility to save time for the tester since defining the expected results of each test case in most cases needs to be done manually and is therefore quite costly.

The test suite is eventually executed, which is step six in our process. While executing the test cases, different types of dynamic information is be collected. The pass-fail result of each test case is an obvious example. The collected information from executing one or more test cases is analyzed in step seven, the last step of our process. Via a second feedback loop, this evaluation step allows the tester to go back to previous steps of the test process to enhance the input parameter model or the combination strategy selection should the results be unsatisfactory.

Introducing the possibility to evaluate the test suite prior to execution have several reasons. As discussed in sections 3.2 and 3.3 the tester makes a number of choices that will determine the contents of the final test suite. The choice of combination strategy and the input parameter modeling are obvious examples. In some cases it is possible to exactly determine the number of test cases that will be the result of applying a certain combination strategy on an input parameter model. However, some of the combination strategies contain an element of randomness built into their algorithms, which makes it difficult to determine the number of test cases in advance. One example is OA [Man85] where the final result, i.e., number of test cases may be affected by the order of the parameters in the test problem. Another example is AETG [CDPP96] which in each step of the algorithm computes a number of test case candidates based to some extent on random values. The "best" candidate is then chosen and the algorithm proceeds to the next iterations. If more that one candidate is "best" then this choice is also made randomly.

Some combination strategies, e.g., AETG [CDPP96] and IPO [LT01] allow the tester to specify

some test cases in advance that must be included in the final test suite. Starting from a predetermined set of test cases, adds to the difficulty of computing the size of the test suite a priori. These examples are arguments for presenting the possibility of evaluation of the test suite to the tester.

Another argument is provided by Williams and Probert [WP01] when they advocate the evaluation of a test suite not originally intended for a specific coverage criterion, e.g., a regression test suite. Such an evaluation may give the tester indications if a test suite is sufficiently effective. This is supported by recent results by Kuhn and Reilly [KR02]. Their results indicate, for sufficiently large values of t, that satisfying t-wise coverage is nearly as effective as satisfying $t+1$-wise coverage. Further, the specific value of t where diminishing returns start to appear differ between $4-6$. Thus, there are means of evaluating both the efficiency, e.g., number of test cases, and effectiveness, e.g., achieved coverage, of a test suite prior to execution.

Dalal, Jain, Karunanithi, Leaton, Lott, Patton, and Horowitz [DJK+99] provide a different argument for the ability to examine the generated test suite, as they claim that creating an appropriate input parameter model often is an iterative process. An input parameter model is created and evaluated and based on the results of the evaluation the input parameter model is modified and re-evaluated and so on until the desired properties of the input parameter model is achieved. From a testing efficiency perspective we believe static analysis prior to test execution is far better than having to implement and execute the test cases in order to determine if an input parameter model is appropriate.

Evaluation of the test suite is not restricted to static properties of the test suite. Piwowarski, Ohba, and Caruso [POC93] show that measuring code coverage may be feasible as late in the test process as during function testing. The achieved code coverage when executing a test suite is used by several authors to evaluate combination strategies [YLDM97, BY98, GLOA03]. Further, Dunietz et al. [DES+97] show the correspondence between t-wise coverage and code coverage. The latter results are more general than a single investigation and thus, more interesting from an evaluation perspective. Thus, the achieved code coverage by executing the test inputs on the test objects can also be used to determine the usefulness of the test suite and hence the initial choice of combination strategies and input parameter model.

Although the suggested combination strategies testing process is prepared for an iterative development of the input parameter model, this research project does not require examination of the effects, in terms of efficiency and effectiveness, of such an approach.

5.2 Combination Strategy Selection

One of the major contributions of this research project is an account of the properties important to consider when selecting combination strategies. For each property we expect to present some method of comparing the different combination strategies. Together these properties and their corresponding assessment methods form a decision support for the tester when deciding which combination strategies to apply to a specific test problem.

As previously mentioned some preliminary comparison information already exists. In sec-

tion 5.2.1 we show how different coverage criteria are related including some combination strategies that fulfill these coverage criteria. In section 5.2.2 the size of the generated test suites as functions of the number of parameters and the number of values of each parameter are presented for some of the combination strategies.

We expect results of similar fashion for the rest of the properties as well.

5.2.1 Subsumption

A direct result of working with coverage criteria associated to combination strategies is the subsumption hierarchy shown in figure 4. The definition of subsumption is from Rapps and Weyuker [RW85]: coverage criterion X *subsumes* coverage criterion Y iff 100% X coverage implies 100% Y coverage (Rapps and Weyukers' original paper used the term inclusion instead of subsumption). We have included all known coverage criteria and in the light of subsumption tried to generalize them.

The left, boxed-in, column of coverage criteria in Figure 4 represent criteria that do not use semantic information. N-wise coverage requires that every combination of all parameter values are included in the test suite. "t-wise coverage" is used as a short form for every level of coverage from $N-1$ down to 2. It is straightforward to see that j-wise coverage subsumes $j-1$-wise coverage, for any j such that $2 \leq j \leq N$.

The right column contains the coverage criteria based only on valid values. For the same reason that j-wise coverage subsumes $j-1$-wise coverage, it is easy to see that the j-wise **valid** coverage criterion subsumes the $j-1$-wise valid coverage criterion.

Note that i-wise coverage where $1 \leq i \leq N-1$ does *not* subsume i-wise valid coverage. This is easily demonstrated for the case where $i = 2$. Consider a three parameter problem, each with one valid value v and one erroneous value e. The following four combinations satisfy pair-wise coverage since each possible pair of values are included $[(v, v, e),\ (v, e, v),\ (e, v, v),\ (e, e, e)]$. However, pair-wise valid coverage is not satisfied since a pair of two valid parameters only count if the remaining parameters also are valid. Since every combination in the example contains at least one erroneous value, the achieved pair-wise valid coverage is 0%. N-wise coverage contains every possible combination so by definition it subsumes all other coverage criteria, including N-wise valid coverage.

The single error coverage criterion represents a different class of coverage criteria. Although it is possible to generalize this idea into multiple error coverage this would introduce the possibility of masking so is omitted. At first it may seem that the single error coverage criterion would subsume the each-used coverage criterion. However, a counter example can be constructed. Consider a two-parameter problem where the first parameter has two valid values (v11 and v12) and one erroneous value (e11). The second parameter has one valid value (v21) and one erroneous value (e21). The two test cases $[(e11, v21),\ (v11, e21)]$ satisfy the single error coverage criterion but not the each-used coverage criterion.

An interesting property of base choice coverage, when assuming that all base choices are valid,

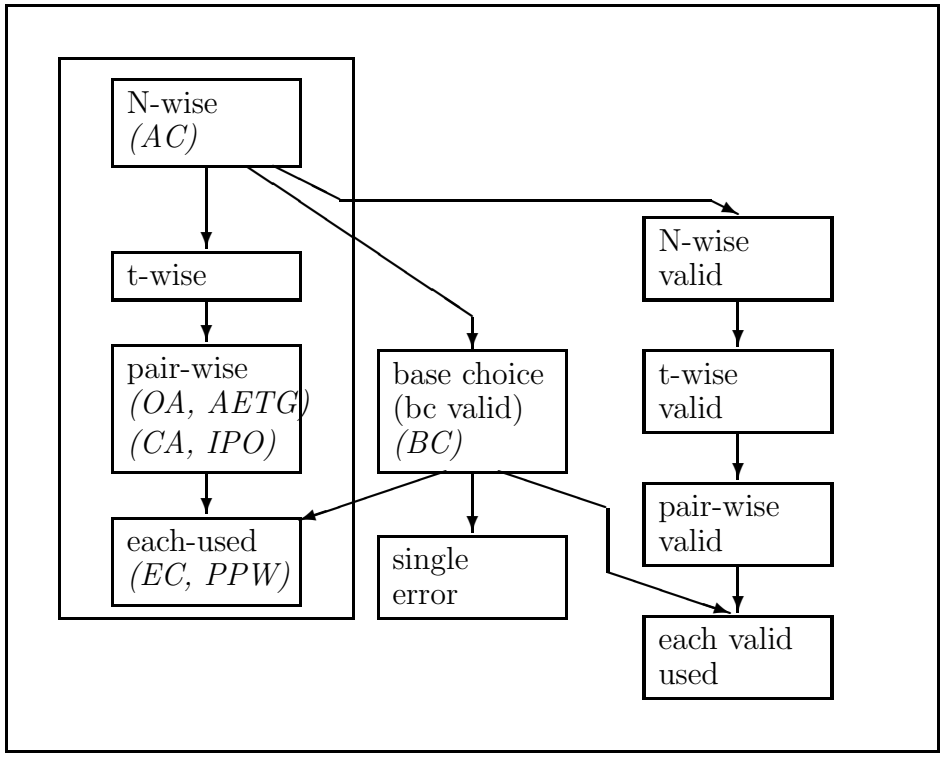

Figure 4: Subsumption hierarchy for the coverage criteria related to combination strategies. Italics indicate combination strategies.

is that it subsumes the each-used, each valid used, and single error coverage criteria. By definition, the base test case contains only valid values. From the definition of the base choice algorithm we get that all test cases differs from the base test case by the value of only one parameter. Further we know that each interesting value of every parameter is included in some test case. Thus, the each-used criterion is subsumed. Further, every valid value of every parameter must appear in at least one test case in which the rest of the values are also valid, since the base test case contains only valid values. Thus, the each valid used criterion is subsumed. Finally, for the same reason, every invalid value of every parameter must occur in exactly one test case in which the rest of the values are valid. Thus, the single error criterion is subsumed.

Combination Strategy	**Test Suite Size**	**Example 1** $N = 8, V_i = 4$	**Example 2** $N = 4, V_i = 8$
AETG	$\sim V_{max}^2$	~ 16	~ 64
OA	$\sim V_{max}^2$	~ 16	~ 64
CA	$\sim N^2 + V_{max} log^2 V_{max}$	~ 65	~ 23
EC	V_{max}	4	8
BC	$1 + \sum(V_i - 1)$	25	29
AC	$\prod V_i$	65536	4096
IPO	$\sim V_{max}^2$	~ 16	~ 64

Table 3: Definite or approximate ($\sim$) test suite sizes for a problem with N parameters and V_i values of the i:th parameter. V_{max} is the largest V_i.

5.2.2 Size of Generated Test Suite

Table 3 gives an overview of the sizes of the generated test suites. In some cases only approximate values are possible to give. The main reason for this is that those algorithms contain some degree of randomness why using the strategy on a test problem may yield different solutions. Also, only a subset of the combination strategies from appendix A is included in the table. There are several reasons. Some strategies, i.e., Rand and AntiRand, have no natural limit. Other strategies, i.e., k-bound and k-perim, differ greatly in the size of the generated test suite depending on the value of k. The algorithm of PPW is not described and finally CATS, having an element of randomness, is not investigated well enough to be described in terms of size of generated test suite.

In addition to the formulas relating the size of the test problem to the size of the generated test suite, table 3 contains two example test problems one with few parameters and many values and one with many parameters with few values.

5.3 Input Parameter Modeling

An important result of our research is a comprehensive method for creating an input parameter model, based on some specification. Our current assumption is that this method will be based on the Category Partition method [OB88]. The method will both a work instruction and a detailed specification of how syntax and semantics of the input parameter model.

Our current proposition is that the input parameter model will consist of four different parts. The first part, the *abstract parameter representation*, only describes the size of the test problem, free of semantic information. The second part, the *semantic model*, will contain both the semantic information needed by the combination strategies to generate the abstract test cases, and the information needed to translate these test cases into real test cases that may possibly be auto-

matically executed. The third part of the input, the *constraint model*, will contain information about subcombinations of parameter values that must not be included in the final test suite. Two types of constraints should be possible to express. Consider parameters $p1$ and $p2$, with interesting values $p1 = \{v11, v12\}$ and $p2 = \{v21, v22, v23, v24\}$. An *incompatibility* constraint is that $v11$ cannot be used with $v21$. A *limiting* constraint is that $v12$ can only be used with elements from the set $\{v23, v24\}$.

These two types of constraints are in some sense equivalent since an incompatible constraint can be rewritten into a set of limiting constraints and vice versa. However, to make it convenient for the tester to express constraints both types should be included in the constraint model.

The fourth part of the input parameter model will contain already existing test cases that should be included in the final test suite.

The different parts of the input parameter model will impose different requirements on the language. The abstract parameter representation will obviously need to contain support for representation of parameters and parameter values. The semantic model will, for instance need to support the representation of base choice values of each parameter. The constraint model needs at least to be able to express exclusion and inclusion of specific and sets of subcombinations.

5.4 Efficiency and Effectivity of Combination Strategies

5.4.1 State-of-Practice

An interesting result of this research proposal is a report on the state-of-practice with respect to how test cases are identified and the cost of testing. This type of information is valuable to the research community since it may help bridge the gap between theory and practice. Thus, this results, is in its own an important contribution.

5.4.2 Cost Model for the Combination Strategies Testing Process

A concrete result of this research project is the cost model developed for estimation of the time consumption of the combination strategies testing process. Within this project it will be used to compare the use of combination strategies with the time consumption of the currently used methods. However, the usage of the cost model is not limited to this. The cost model may also be used to tune the usage of the combination strategies testing process. Future extensions of the model may also be used when planning a test project to make more realistic estimations of resource usage.

5.4.3 Proof-of-Concept

The proof-of-concept, even if we get a negative answer to our research question, will yield important knowledge in how testing is performed in practice. Should we get a positive result, it would be of

interest both to the research community and to the practicing testers to widen the research of the applicability of combination strategies.

6 Related Work

All of the work relating to this can be found in appendices A and B. However, the set of papers by Cohen, Dalal et al. [CDFP97, DJK+98, DJK+99, DJP+99] deserves special attention. Their work has resulted in a commercial tool called $AETG^{SM}$ which is used to generate a set of test case inputs based on the information in an input parameter model. This is very related to the intentions of this research project. Thus, we want to point at some crucial differences. First of all we do not limit ourselves to the AETG tool or its algorithms. Instead we include the selection of combination strategies to use in our process. Second and even more important, our aim is to describe, in a laymen manner how to create an input parameter model. In the case of the AETG tool, little information is given on how to arrive at a suitable input parameter model. A third difference is that we intend to involve laymen testers to validate our hypothesis that combination strategies are feasible alternatives for the practicing tester.

7 Conclusions

This section concludes this thesis proposal by giving a short summary of the project, its contributions, and some ideas of future work.

7.1 Summary

Based on the existing work on combination strategies we have proposed a research project aiming to find out if combination strategies are feasible alternatives in practical testing. The starting point of this research project is the definition of a combination strategies testing process. The main aim is to describe each step in the process in such detail that the complete process may be used in a real testing project. The central tasks of the process, and also this research, are combination strategy selection and input parameter modeling. Methods to perform these tasks will be developed and evaluated separately and in the scope of the testing process.

State-of-practice of used testing case selection methods and their performance will be investigated and compared to results of using the combination strategies testing process. If the combination strategies testing process is either more effective in finding faults or more cost-effective (efficient) in finding faults we conclude that combination strategies are feasible alternatives to the currently used test case selection methods.

This report defines a number of research objectives that will lead to the answering of our main research question. For each research objective, the approaches to reach the objective are described.

These methods include literature surveys, experiments, tool implementations, interviews, and a final proof-of-concept, in which the complete process is tested and evaluated in a case study.

7.2 Contributions

The main contribution of this research is an increased understanding of the advantages and limitations of combination strategies, in particular with respect to practical testing. A number of methods and metrics for and results from comparisons of different combination strategies will be one part of this increased understanding. A testing process tailored for the use of combination strategies, including a set of work instructions, is another part.

A set of tools that automate some of the steps in the test process is also part of the contribution. Within this area not only the tools are important but also languages and formats of intermediate representations of the test case information. For instance the way to represent the input parameter model is of crucial importance for the final result.

Last, but not least, this research will yield better knowledge of the state-of-practice in industry with respect to used test case selection methods and their true performance.

7.3 Future Work

An interesting expansion of this work is to examine ways of making the combination strategies testing process more efficient. For instance, the feed-back loop from the evaluation step back to the combination strategy selection and input parameter modeling is one area where more work can be performed. Another, related area, which would be interesting to investigate is how other test case selection methods would behave in the context to the suggested process.

It is likely that an optimal test result cannot be achieved with one single test case selection method. Using several test case selection methods together will probably be needed. A big research challenge is to find out how to identify a good selection of test case selection methods.

Another interesting area, which may be seen as an extension of this work is to further explore the possibilities of using combination strategies in testing. As been stated previously, ideas on how to use combination strategies to solve several different test problems already exist, e.g. unit testing, object oriented testing, functional testing, feature interaction testing, robustness testing, etc. It is likely that combination strategies are useful in yet other areas of testing, for instance real-time testing, load and performance testing etc.

8 Acknowledgments

I would like to thank my supervisor Sten F. Andler and my co-supervisors A. Jefferson Offutt and Mariam Kamkar, whose help has been of decisive importance for the creation of this document. My mentors Jonas Mellin and Thomas Vesterlund have also made a number of important contributions

along the way of formulating and writing this document. The rest of the Distributed Real-Time Systems group, in particular Birgitta Lindström and Robert Nilsson have also played important roles as discussion partners, reviewers, and general motivators. Some of my colleagues at Enea Systems AB, for instance Maria Jönsson, Krister Ström, and Anders Claesson have also been great sources of inspirations Last, but not least I wish to express my gratitude towards my sponsors, Enea Systems AB and KK-stiftelsen who are generously supporting this research.

9 Bibliography

References

[AO94] Paul E. Ammann and A. Jefferson Offutt. Using formal methods to derive test frames in category-partition testing. In *Proceedings of the Ninth Annual Conference on Computer Assurance (COMPASS'94),Gaithersburg MD*, pages 69–80. IEEE Computer Society Press, June 1994.

[Bei90] Boris Beizer. *Software Testing Techniques*. Van Nostrand Reinhold, 1990.

[BJE94] Kirk Burroughs, Aridaman Jain, and Robert L. Erickson. Improved quality of protocol testing through techniques of experimental design. In *Proceedings of the IEEE International Conference on Communications (Supercomm/ICC'94), May 1-5, New Orleans, Louisiana, USA*, pages 745–752. IEEE, May 1994.

[BPP92] Robert Brownlie, James Prowse, and Mahdav S. Phadke. Robust Testing of AT&T PMX/StarMAIL using OATS. *AT&T Technical Journal*, pages 41–47, May/June 1992.

[BR03] Tomas Berling and Per Runesson. Efficient Evaluation of Multi-Factor Dependent System Performance using Fractional Factorial Design. *IEEE Transactions on Software Engineering*, 29(9):769–781, October 2003.

[BS87] Victor R. Basili and Richard W. Selby. Comparing the Effectiveness of Software Testing Strategies. *IEEE Transactions on Software Engineering*, SE-13(12):1278–1296, December 1987.

[BS97] R. Biyani and P. Santhanam. TOFU: Test Optimizer of Functional Usage. Technical Report Software Engineering Technical Brief 2(1), IBM T.J. Watson Research Center, 1997.

[BY98] Kevin Burr and William Young. Combinatorial test techniques: Table-based automation, test generatio and code coverage. In *Proceedings of the International Conference on Software Testing, Analysis, and Review (STAR'98), San Diego, CA, USA, October 26-28, 1998*, 1998.

[CDFP97] David M. Cohen, Siddhartha R. Dalal, Michael L. Fredman, and Gardner C. Patton. The AETG System: An Approach to Testing Based on Combinatorial Design. *IEEE Transactions on Software Engineering*, 23(7):437–444, July 1997.

[CDKP94] 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.

[CDPP96] David M. Cohen, Siddhartha R. Dalal, Jesse Parelius, and Gardner C. Patton. The Combinatorial Design Approach to Automatic Test Generation. *IEEE Software*, pages 83–88, September 1996.

[DES+97] I. S. Dunietz, W. K. Ehrlich, B. D. Szablak, C. L. Mallows, and A. Iannino. Applying design of experiments to software testing. In *Proceedings of 19th International Conference on Software Engineering (ICSE'97), Boston, MA, USA 1997*, pages 205–215. ACM, May 1997.

[DHS02] Nigel Daley, Daniel Hoffman, and Paul Strooper. A framework for table driven testing of Java classes. *Software - Practice and Experience (SP&E)*, 32:465–493, 2002.

[DJK+98] S.R. Dalal, A. Jain, N. Karunanithi, J.M. Leaton, and C.M. Lott. Model-based testing of a highly programmable system. In *Proceedings of 9th International Symposium in Software Engineering (ISSRE'98) 1998, Paderborn, Germany, 4-7 November 1998*, pages 174–178. IEEE Computer Society Press, 1998.

[DJK+99] S.R. Dalal, A. Jain, N. Karunanithi, J.M. Leaton, C.M. Lott, G.C. Patton, and B.M. Horowitz. Model-based testing in practice. In *Proceedings of 21st International Conference on Software Engineering (ICSE'99) 1999, Los Angeles, CA, USA, 16-22 May 1999*, pages 285–294. ACM Press, 1999.

[DJP+99] Siddharta R. Dalal, Ashish Jain, Gardner Patton, Manish Rathi, and Paul Seymour. AETG web: A web based service for automatic efficient test generation from functional requirements. In *Proceedings of 2nd IEEE Workshop on Industrial-Strength Formal Specification Techniques, Boca Raton, Florida, USA, October 21-23, 1999*, pages 84–85. IEEE Computer Society, 1999.

[DLS78] Richard A. DeMillo, Richard J. Lipton, and Frederick G. Sayward. Hints on Test Data Selection: Help for the Practicing Programmer. *IEEE Computer*, pages 34–41, apr 1978.

[DM98] Siddharta Dalal and Colin L. Mallows. Factor-Covering Designs for Testing Software. *Technometrics*, 50(3):234–243, August 1998.

[DN84] J. W. Duran and S. C. Ntafos. An Evaluation of Random Testing. *IEEE Transactions on Software Engineering*, pages 438–444, July 1984.

[GG93] Matthias Grochtmann and Klaus Grimm. Classification Trees for Partition Testing. *Journal of Software Testing, Verification, and Reliability*, 3(2):63–82, 1993.

[GLOA03] Mats Grindal, Birgitta Lindström, A. Jefferson Offutt, and Sten F. Andler. An Evaluation of Combination Strategies for Test Case Selection, Technical Report. Technical Report HS-IDA-TR-03-001, Department of Computer Science, University of Skövde, 2003.

[Gut99] Walter J. Gutjahr. Partition Testing vs. Random Testing: The Influence of Uncertainty. *IEEE Transactions on Software Engineering*, 25(5):661–674, September/October 1999.

[Hel95] E. Heller. Using design of experiment structures to generate software test cases. In *Proceedings of the 12th International Conference on Testing Computer Software, New York, USA, 1995*, pages 33–41. ACM, 1995.

[HSW99] D.M. Hoffman, P.A. Strooper, and L. White. Boundary Values and Automated Component Testing. *Journal of Software Testing, Verification, and Reliability*, 9(1):3–26, 1999.

[HT90] Dick Hamlet and Ross Taylor. Partition Testing Does Not Inspire Confidence. *IEEE Transactions on Software Engineering*, 16(12):1402–1411, December 1990.

[Hul00] Jerry Huller. Reducing time to market with combinatorial design method testing. In *Proceedings of 10th Annual International Council on Systems Engineering (INCOSE'00) 2000, Minneapolis, MN, USA, 16-20 July 2000*, 2000.

[KKS98] Nathan P. Kropp, Philip J. Koopman, and Daniel P. Siewiorek. Automated robustness testing of off-the-shelf software components. In *Proceedings of FTCS'98: Fault Tolerant Computing Symposium, June 23-25, 1998 in Munich, Germany*, pages 230–239. IEEE, 1998.

[KR02] D. Richard Kuhn and Michael J. Reilly. An investigation of the applicability of design of experiments to software testing. In *Proceedings of the 27th NASA/IEE Software Engineering Workshop, NASA Goodard Space Flight Center, MD, USA, 4-6 December, 2002*. NASA/IEEE, December 2002.

[LT98] Yu Lei and K. C. Tai. In-parameter-order: A test generation strategy for pair-wise testing. In *Proceedings of the third IEEE High Assurance Systems Engineering Symposium*, pages 254–261. IEEE, November 1998.

[LT01] Yu Lei and K. C. Tai. A Test Generation Strategy for Pairwise Testing. Technical Report TR-2001-03, Department of Computer Science, North Carolina State University, Raleigh, 2001.

[Mal95] Yashwant 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, October 1995.

[Man85] Robert Mandl. Orthogonal Latin Squares: An application of experiment design to compiler testing. *Communications of the ACM*, 28(10):1054–1058, October 1985.

[Mye78] Glenford J. Myers. A Controlled Experiment in Program Testing and Code Walkthroughs/Inspection. *Communcations of the ACM*, 21(9):760–768, September 1978.

[Mye79] Glenford J. Myers. *The art of Software Testing.* John Wiley and Sons, 1979.

[OB88] Thomas J. Ostrand and Marc J. Balcer. The Category-Partition Method for Specifying and Generating Functional Tests. *Communications of the ACM*, 31:676–686, June 1988.

[POC93] Paul Piwowarski, Mitsuru Ohba, and Joe Caruso. Coverge measure experience during function test. In *Proceedings of 14th International Conference on Software Engineering (ICSE'93), Los Alamitos, CA, USA 1993*, pages 287–301. ACM, May 1993.

[Rei97] Stuart Reid. An empirical analysis of equivalence partitioning, boundary value analysis and random testing. In *Proceedings of the 4th International Software Metrics Symposium (METRICS'97), Albaquerque, New Mexico, USA, Nov 5-7, 1997*, pages 64–73. IEEE, 1997.

[RW85] Sandra Rapps and Elaine J. Weyuker. Selecting Software Test Data Using Data Flow Information. *IEEE Transactions on Software Engineering*, SE-11:367–375, april 1985.

[SCSK02] Sun Sup So, Sung Deok Cha, Timothy J. Shimeall, and Yong Rae Kwon. An empirical evaluation of six methods to detect faults in software. *Software Testing, Verification and Reliability*, 12(3):155–171, September 2002.

[She94] G. Sherwood. Effective testing of factor combinations. In *Proceedings of the third International Conference on Software Testing, Analysis, and Review (STAR94), Washington DC.* Software Quality Eng., May 1994.

[Tag87] G. Taguchi. *System of Experimental Design.* ed. D. Clausing, vols 1 and 2, UNIPUB/Kraus International Publications, New York, 1987.

[TKG+90] J. Toczki, F. Kocsis, T. Gyimóthy, G. Dányi, and G. Kókai. Sys/3-a software development tool. In *Proceedings of the third international workshop of compiler compilers (CC'90), Schwerin, Germany, October 22-24, 1990*, pages 193–207. Springer Verlag, Lecture Notes in Computer Science (LNCS 477), 1990.

[TL02] Kuo-Chung Tai and Yu Lei. A Test Generation Strategy for Pairwise Testing. *IEEE Transactions on Software Engineering*, 28(1):109–111, January 2002.

[Wil00] Alan W. Williams. Determination of test configurations for pair-wise interaction coverage. In *Proceedings of the 13th International Conference on the Testing of Communicating Systems (TestCom 2000), Ottawa, Canada, August 2000*, pages 59–74, August 2000.

[WP96] Alan W. Williams and Robert L. Probert. A practical strategy for testing pair-wise coverage of network interfaces. In *Proceedings of the 7th International Symposium on Software Reliability Engineering (ISSRE96), White Plains, New York, USA, Oct 30 - Nov 2, 1996*, Nov 1996.

[WP01] Alan W. Williams and Robert L. Probert. A measure for component interaction test coverage. In *Proceedings of the ACSI/IEEE International Conference on Computer Systems and Applications (AICCSA 2001), Beirut, Lebanon, June 2001*, pages 304–311, June 2001.

[WP02] Alan W. Williams and Robert L. Probert. Formulation of the interaction test coverage problem as an integer problem. In *Proceedings of the 14th International Conference on the Testing of Communicating Systems (TestCom 2002), Berlin, Germany, March 2002*, pages 283–298, March 2002.

[WRBM97] M. Wood, M. Roper, A. Brooks, and J. Miller. Comparing and combining software defect detection techniques: A replicated study. In *Proceedings of the Sixth European Software Engineerng Conference (ESEC/FSE 97)*, pages 262–277. Springer Verlag, Lecture Notes in Computer Science Nr. 1013, September 1997.

[YLDM97] Huifang Yin, Zemen Lebne-Dengel, and Yashwant K. Malayia. Automatic Test Generation using Checkpoint Encoding and Antirandom Testing. Technical Report CS-97-116, Colorado State University, 1997.

A Combination Strategies - Methods

This section briefly describes the combination strategies that have been identified in the literature. The combination strategies have been organized into different classes based on the amount of randomness of the algorithm and according to how the test suites are created. Figure 5 shows an overview of the classification scheme. The combination strategies labeled *non-deterministic* all depend to some degree on randomness. A property of these combination strategies is that the same input parameter model may lead to different test suites. The simplest non-deterministic combination strategy is pure *random* selection of test cases. The group of non-deterministic combination strategies also include two heuristic methods, *CATS* and *AETG*.

The *deterministic* combination strategies group is further divided into three subgroups, *instant*, *iterative*, and *parameter-based*. All of these combination strategies will always produce the same result from a specific input parameter model. The two instant combination strategies, *Orthogonal Arrays (OA)* and *Covering Arrays (CA)*, produce the complete test suite directly. The largest group of combination strategies is *iterative*. They share the property that the algorithms generate one test case at a time and adds it to the test suite. *Each Choice (EC)*, *Partly Pair-Wise (PPW)*, *Base Choice (BC)*, *All Combinations (AC)*, and *Anti-random (AR)* all belong to the iterative combination strategies. The parameter-based combination strategy, *In Parameter Order (IPO)*, starts by creating a test suite for a subset of the parameters in the input parameter model. Then one parameter at a time is added and the test cases in the test suite are modified to cover the new parameter. Completely new test cases may also need to be added.

The third main group of combination strategies, *compound*, include all strategies in which two or more combination strategies are used together. Two such strategies from the literature are used to illustrate the concept.

A.1 Non-Deterministic Combination Strategies

Non-deterministic combination strategies all share the property that chance will play a certain role in determining which tests are generated. This means two executions of the same algorithm with the same preconditions may produce different results.

A.1.1 Heuristic Combination Strategies

Heuristic t-wise (CATS)

The CATS tool for generating test cases is based on a heuristic algorithm that can be custom designed to satisfy t-wise coverage. The algorithm was described by Sherwood [She94] and a version of the algorithm that generates a test suite to satisfy pair-wise coverage is shown in Figure 6.

The heuristic nature of the algorithm makes it impossible to exactly calculate the number of test cases in a test suite generated by the algorithm.

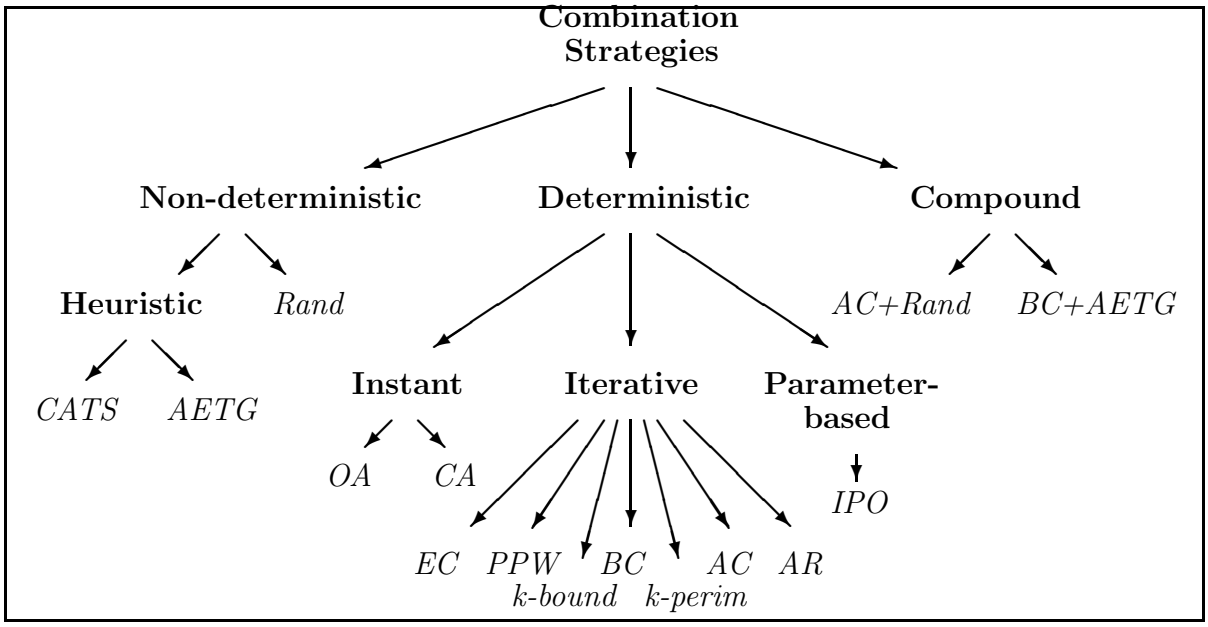

Figure 5: Classification Scheme for Combination Strategies

```
Assume test cases t_1 − t_{i−1} already selected
Let Q be the set of all possible combinations not yet selected
Let UC be a set of all pairs of values of any two parameters that are
not yet covered by the test cases t_1 − t_{i−1}

A) Select t_i by finding the combination that covers most pairs in UC.
   If more than one combinations covers the same amount select the
     first one encountered.
   Remove the selected combination from Q.
   Remove the covered pairs from UC.
B) Repeat until UC is empty.
```

Figure 6: CATS heuristic algorithm for achieving pair-wise coverage

```
Assume test cases t_1 − t_{i−1} already selected
Let UC be a set of all pairs of values of any two parameters that are
not yet covered by the test cases t_1 − t_{i−1}

A) Select candidates for t_i by
     1) Selecting the variable and the value included in most
        pairs in UC.
     2) Making a random order of the rest of the variables.
     3) For each variable, in the sequence determined by step two,
        select the value included in most pairs in UC.
B) Repeat steps 1-3 k times and let t_i be the test
   case that covers the most pairs in UC. Remove those pairs from UC.

Repeat until UC is empty.
```

Figure 7: AETG heuristic algorithm for achieving pair-wise coverage

Heuristic t-wise (AETG)

Burroughs, Jain, and Erickson [BJE94] and Cohen, Dalal, Kajla, and Patton [CDKP94] report on the use of a tool called Automatic Efficient Test Generator (AETG), which contains an algorithm for generating all pair-wise combinations. Cohen et al. [CDPP96, CDFP97] later described a heuristic greedy algorithm for achieving t-wise coverage, where t is an arbitrary number. This algorithm was implemented in the AETG tool. Figure 7 shows an instance of the algorithm that will generate a test suite to satisfy pair-wise coverage.

The number of test cases generated by the algorithm for a specific test problem is related to the number of candidates (k in the algorithm in Figure 7) for each test case. In general, larger values of k yield smaller numbers of test cases. However, Cohen et al. report that using values larger than 50 will not give any dramatic decrease in the number of test cases.

The heuristic nature of the algorithm makes it impossible to theoretically calculate the number of test cases in a test suite generated by the algorithm. However, empirical results show similar behavior as OA, i.e., $O(v_{max}^2$[GLOA03].

A.1.2 Random Combination Strategies

Random (Rand)

Creating a test suite by randomly sampling test cases from the complete set of test cases based on some input distribution (often uniform distribution) is an old idea with unclear origin. Duran and Ntafos [DN84] made an evaluation of random testing in 1984. In the scope of combination

1	2	3
3	1	2
2	3	1

Figure 8: A 3×3 Latin Square

strategies, Mandl [Man85] may have been the first to mention the idea. Mandl states (without motivation) that a reasonable random selection would probably be about the same size as a test suite that simply takes the N variables in turn, and includes for each of them v test cases to cover the v interesting values of the variable. In 1997, Dunietz et al. [DES+97] applied the random combination strategy with varying number of test cases.

A.2 Deterministic Combination Strategies

Deterministic combination strategies share the property that they produce the same test suite every time they are used. These strategies are divided into three subcategories: instant, iterative and parameter-based combination strategies. Instant combination strategies create the complete test suite all at once. Iterative combination strategies build up the test suite by adding one test case at a time based on what has already or not yet been covered. Parameter-based combination strategies build up the test suite by adding values to the test cases for one parameter at a time.

A.2.1 Instant Combination Strategies

Sometimes a tester wants to include her favorite test cases in the test suite. With instant combination strategies this will not affect the final result. The same test suite will be selected by the combination strategy regardless if other some [other] test cases have already been preselected.

Orthogonal Arrays (OA)

Orthogonal Arrays is a mathematical concept that has been known for quite some time. The application of orthogonal arrays to testing was first introduced by Mandl [Man85] and later more thoroughly described by Williams and Probert [WP96].

The foundation of OA is Latin Squares. A *Latin Square* is an $V \times V$ matrix completely filled with symbols from a set that has cardinality V. The matrix has the property that the same symbol occurs exactly once in each row and column. Figure 8 contains an example of a 3×3 Latin Square with the symbols $\{1, 2, 3\}$.

Two Latin Squares are *orthogonal* if, when they are combined entry by entry, each pair of elements occurs precisely once in the combined square. Figure 9 shows an example of two orthogonal 3×3 Latin Squares and the resulting combined square.

If indexes are added to the rows and the columns of the matrix, each position in the matrix

1	2	3
3	1	2
2	3	1

1	2	3
2	3	1
3	1	2

$1,1$	$2,2$	$3,3$
$3,2$	$1,3$	$2,1$
$2,3$	$3,1$	$1,2$

Figure 9: Two orthogonal 3×3 Latin Squares and the resulting combined square

can be described as a tuple $< x, y, z_i >$, where z_i represents the values of the $< x, y >$ position. Figure 10 contains the indexed Latin Square from Figure 8 and Figure 11 contains the resulting set of tuples. The set of all tuples constructed by a Latin Square satisfies pair-wise coverage.

Coordinates	**1**	**2**	**3**
1	1	2	3
2	3	1	2
3	2	3	1

Figure 10: A 3×3 Latin Square augmented with coordinates

tuple	**Tuple** $< xyz >$
1	111
2	123
3	132
4	212
5	221
6	233
7	313
8	322
9	331

Figure 11: Tuples from the 3×3 Latin Square that satisfies pair-wise coverage

To illustrate how orthogonal arrays are used to create test cases, consider a test problem with three parameters a, b, and c, where three values have been chosen as interesting for a, two for b, and two for c. To create test cases from the orthogonal array tuples a mapping between the coordinates of the tuples and the parameters of the test problem must be established. Let each coordinate represent one parameter and the different interesting values of the parameter map to

the different values of that coordinate. In the case of the example, map a onto x, b onto y, and c onto z. This mapping presents no problem for parameter a, but for parameters b and c there are more coordinate values than there are values. To resolve this situation, each test case that has a coordinate value without a corresponding interesting value needs to be changed. Consider tuple 7 in Figure 11. The value of coordinate z is 3 and only values 1 and 2 are defined in the mapping to parameter c. To create a test case from tuple 7, the undefined value should be replaced by an arbitrarily defined value, which in this case is 1 or 2. Sometimes it is possible to replace undefined values in such a way that a test case can be removed. As an example of this consider tuple 6. The values for both y and z are undefined and so should be changed to valid values. Set y to 1 and z to 2 and the changed tuple is identical to tuple 4 and thus is not needed to include in the test suite.

A test suite based on orthogonal arrays satisfies pair-wise coverage, even after undefined values have been replaced and possibly some duplicate tuples have been removed. This means that the approximate number of test cases generated by the orthogonal arrays combination strategy is v_{max}^2, where $v_{max} = Max_{i=1}^{N} v_i$, N is the number of parameters, and v_i is the number of values of parameter i. Williams and Probert [WP96] give further details on how test cases are created from orthogonal arrays.

Sloane has collected a large number of precalculated orthogonal arrays of various sizes and made them available on the Web[2].

Covering Arrays (CA)

Covering Arrays [Wil00] is a refinement of orthogonal arrays. A property of orthogonal arrays is that they are balanced, which means that each parameter value occurs the same number of times in the test suite. If only t-wise (for instance pair-wise) coverage is desired the balance property is unnecessary and will make the algorithm less efficient. In a covering array that satisfies t-wise coverage, each t-tuple occurs at least once but not necessarily the same number of times. Another problem with orthogonal arrays is that for some problem sizes there does not exist enough orthogonal arrays to represent the entire problem. This problem is also avoided by using covering arrays.

A covering array is constructed for a test problem from a set of building blocks:

- $O(k^2,\ k+1,\ k)$ is an orthogonal array with k^2 rows and $k+1$ parameters, which have at most k values each.
- $B(k^2-1,\ k+1,\ k,\ d) = O(k^2,\ k+1,\ k)$ with the first row removed and with the columns used d times each consecutively.
- $R(k^2-k,\ k,\ k,\ d) = O(k^2,\ k+1,\ k)$ with the k first rows and the first column removed and with the remaining columns used d times each consecutively.
- $I(c,\ d)$ is a matrix with c rows and d columns completely filled with "ones."

[2]URL: http://www.research.att.com/~njas/oadir, page visited March 2004.

- $N(k^2 - k,\ k,\ d)$ is a matrix with $k^2 - k$ rows and $k \times d$ columns, which of which consists of $k \times d$ submatrices filled with "twos", "threes", etc. up to k.

Depending on the size of the problem, these five building blocks are combined in different ways to create a test suite with pair-wise coverage over all parameters and parameter values. The number of test cases required to reach pair-wise coverage has a complexity of $N^2 + v_{max} log^2 v_{max}$ where N is the number of parameters and v_{max} is the maximum number of values of any one parameter.

Both the algorithms for finding suitable building blocks and for combining these to fit a specific test problem have been automated.

A.2.2 Iterative Combination Strategies

Iterative combination strategies are those in which one test case at a time is generated added to the test suite. Thus, a tester may start the algorithm with an already preselected set of test cases.

Each Choice (EC)

The basic idea behind the *Each Choice* combination strategy is to include each value of each parameter in at least one test case. This is achieved by generating test cases by successively selecting unused values for each parameter. This strategy was invented by Ammann and Offutt [AO94], who also suggested that EC can result in an undesirable test suite since the test cases are constructed mechanically without considering any semantic information.

It is clear from the definition of EC that it satisfies each-used coverage. Further, the number of test cases in an EC test suite is at least $Max_{i=1}^{N} v_i$, where N is the number of parameters of the test problem and v_i is the number of values of parameter i.

Partly Pair-Wise (PPW)

Burroughs, Jain, and Erickson [BJE94] sketch a "traditional" combination strategy in which all pair-wise combinations of the values of the two most significant parameters should be included, while at the same time including each value of the other parameters at least once. No details of the actual algorithm are given. Thus, it is impossible to assess the size of the generated test suite. However, it is clear that PPW satisfies each-used coverage.

Base Choice (BC)

The *Base Choice* (BC) combination strategy was proposed by Ammann and Offutt [AO94]. The essence of the idea behind BC was later described by Cohen et al. [CDKP94], and called "default testing." The first step of BC is to identify a base test case. The *base test case* combines the most "important" value for each parameter. Importance may be based on any predefined criterion such as most common, simplest, smallest, or first. Ammann and Offutt suggest "most likely used." Similarly, Cohen et al. call these values "default values." From the base test case,

new test cases are created by varying the values of one parameter at a time while keeping the values of the other parameters fixed on the values in the base test case.

A test suite generated by BC satisfies each-used coverage since each value of every parameter is included in the test suite. If "most likely used" is used to identify the base test case, BC also satisfies the single error coverage criterion. The reason is that the base test case only consists of normal values and only one parameter at a time differs from the base test case.

Ammann and Offutt state that satisfying base-choice coverage does not guarantee the adequacy of the test suite for a particular application. However, a strong argument can be made that a test suite that does not satisfy base-choice coverage *is* inadequate.

A BC test suite has $1 + \sum_{i=1}^{N}(v_i - 1)$ test cases where N denotes the number of parameters of the test problem and v_i is the number of values of parameter i.

A variant of BC was proposed by Burr and Young [BY98], which they called *default testing*. In their version all parameters except one contains the default value, and the remaining parameters contain a maximum or a minimum. This variant does not necessarily satisfy each-used coverage.

All Combinations (AC)

The *All Combinations* (AC) combination strategy algorithm generates every combination of values of the different parameters. The origin of this method is impossible to trace back to a specific person due to its simplicity. It is often used as a benchmark with respect to the number of test cases [Man85, AO94, CDFP97].

An AC test suite satisfies N-wise coverage and contains $\prod_{i=1}^{N} v_i$ test cases, where N denotes the number of parameters of the test problem and v_i is the number of values of parameter i.

Antirandom (AR)

Antirandom testing was proposed by Malayia [Mal95], and is based on the idea that test cases should be selected to have maximum "distance" from each other. Parameters and interesting values of the test object are encoded using a binary vector such that each interesting value of every parameter is represented by one or more binary values. Test cases are then selected such that a new test case resides on maximum Hamming or Cartesian distance from the already selected test cases. In the original description, the antirandom testing scheme is used to create a total order among all the possible test cases rather than attempting to limit the number of test cases. However, just like it is possible in random testing to set a limit to how many test cases that should be generated, the same applies to antirandom test cases. Antirandom testing can be used to select a subset of all possible test cases, while ensuring that they are as far apart as possible.

k-boundary (k-bound) and k-perimeter (k-perim)

With a single parameter restricted to values $[L, .., U]$, generating boundary values is easy. The tester can select the set $\{L,\ U\}$ or $\{L,\ L{+}1,\ U{-}1,\ U\}$. With multiple domains two problems arise: (1) the number of points grows rapidly and (2) there are several possible interpretations. Hoffman, Strooper and White [HSW99] defined the k-boundary and k-perimeter combination strategies to

handle these problems.

The *1-boundary* of an individual domain is the smallest and largest values of that domain (L and U). The *2-boundary* of an individual domain is the next to the smallest and the next to the largest values of that domain and so on ($L + 1$ and $U - 1$).

The *k-boundary* of a set of domains is the Cartesian product of the k-boundaries of the individual domains. Thus, the number of test cases of the k-boundary is 2^N, where N is the number of domains. In the general case, k-boundary does not satisfy any of the normal coverage criteria associated with combination strategies.

The *k-perimeter* test suite can be constructed by (1) including the k-boundary test suite, (2) forming all possible pairs of test cases from the k-boundary test suite such that the two test cases in a pair only differ in one dimension, e.g., ($L,\ L,\ L,\ L$) and ($U,\ L,\ L,\ L$), (3) For each identified pair of test cases, adding to the k-perimeter test suite all points in between the two test cases of the pair i.e., $\{(L+1,\ L,\ L,\ L), (L+2,\ L,\ L,\ L), ..., (U-2,\ L,\ L,\ L), (U-1,\ L,\ L,\ L)\}$.

k-perimeter does not satisfy any coverage criterion in the general case, however, 1-perimeter satisfies each-used coverage.

A.2.3 Parameter-Based Combination Strategies

There is only one parameter-based combination strategy, *In Parameter Order (IPO)*.

In Parameter Order (IPO)
For a system with two or more parameters, the in-parameter-order (IPO) combination strategy [LT98, LT01, TL02] generates a test suite that satisfies pair-wise coverage for the values of the first two parameters. The test suite is then extended to satisfy pair-wise coverage for the values of the first three parameters, and continues to do so for the values of each additional parameter until all parameters are included in the test suite.

To extend the test suite with the values of the next parameter, the IPO strategy uses two algorithms. The first algorithm, horizontal growth, as shown in figure 12, extends the existing test cases in the test suite with values of the next parameter The second algorithm, vertical growth, as shown in figure 13, creates additional test cases such that the test suite satisfies pair-wise coverage for the values of the new parameter.

The nature of the IPO algorithms makes it difficult to theoretically calculate the number of test cases in a test suite generated by the algorithm. An algorithm that closely resembles the IPO algorithm was informally described by Huller [Hul00].

A.3 Compound Combination Strategies

When two or more combination strategies are used together they form a *compound combination strategy*. From the literature we have taken two examples to illustrate this.

Algorithm $IPO\_H\ (\tau, P_i)$
{
  Let τ be a test suite that satisfies pair-wise coverage for the values of parameters p_1 to p_{i-1}.
  Assume that parameter p_i contains the values $v_1, v_2, ..., v_q$
  $\pi = \{$ pairs between values of p_i and values of p_1 to $p_{i-1}\}$
  if $(|\ \tau\ | \leq q)$
  {
    for $1 \leq j \leq |\ \tau\ |$, extend the jth test in τ by adding value v_j and remove from π pairs covered by the extended test.
  }
  else
  {
    for $1 \leq j \leq q$, extend the jth test in τ by adding value v_j and remove from π pairs covered by the extended test
    for $q < j \leq |\ \tau\ |$, extend the jth test in τ by adding one value of p_i such that the resulting test covers the most number of pairs in π, and remove from π pairs covered by the extended test
  }
}

Figure 12: IPO_H – An algorithm for horizontal growth of a test suite by adding values for new parameters

```
Algorithm IPO_V (τ, π)
{
  Let τ′ be an empty set
  for each pair in π
  { assume that the pair contains value w of p_k, 1 ≤ k < i, and value u of p_i
  {
    if (τ′ contains a test case with ‘‘-’’ as the value of p_k
    and u and the value of p_i)
        modify this test case by replacing the ‘‘-’’ with w
  else
        add a new test case to τ′ that has w as the value of p_k, u as
        the value of p_i, and ‘‘-’’ as the value of every other parameter;
  }
  τ = τ ∪ τ′
}
```

Figure 13: IPO_V – An algorithm for vertical growth of a test suite by adding values for new parameters

All or Random (ACoRand)

One problem with the AC strategy is that it may generate infeasibly large test suites. For test problems with less that Y possible test cases the *All or Random* compound combination strategy generates all combinations of parameter values. If the number of possible test cases exceed Y, exactly Y test cases are (deterministically) pseudo-randomly selected. Kropp, Koopman, and Siewiorek [KKS98] suggest 5000 as a suitable value for Y.

An ACoRand test suite contains $t = \prod_{i=1}^{N} v_i$ tests if $t \leq Y$ else Y test cases, where N denotes the number of parameters of the test problem, v_i is the number of values of parameter i, and Y is a predefined maximum number of test cases.

Base Choice and AETG (BCAETG)

In a comparative study, Grindal, Lindström, Offutt, and Andler [GLOA03] show that the union of the test suites generated by BC and AETG is more effective in finding faults than each one individually. A more efficient alternative than taking the union may be to use the BC test suite as input to the generation of the AETG test suite.

B Combination Strategies – Experience and Evaluations

This section contains brief descriptions of the reported results relating to combination strategies. The papers described in this section are organized into eight groups according to their main focus (the combination strategy) in each paper. Within each group the papers are presented in chronological order. While we considered organizing all of the papers in chronological order, the number of papers (over 30!) convinced us that some sort of categorization was needed.

B.1 General Focus

The five papers in this group considered aspects that are general to all combination strategies.

B.1.1 Piwowarski, Ohba and Caruso

Piwowarski, Ohba, and Caruso [POC93] described how to successfully apply code coverage as a stopping criterion during functional testing. The authors formulated functional testing as a problem of selecting test cases from a set of possible test cases made up of all combinations of values of the input parameters. No algorithm was given to choose from this sample, but the authors discuss several methods informally. One method is to fix some parameter values based on some equivalence partition assumption. Another method is by random selection.

B.1.2 Dunietz, Ehrlich, Szablak, Mallows and Iannino

Dunietz, Ehrlich, Szablak, Mallows, and Iannino [DES+97] examined the correlation between t-wise coverage and achieved code coverage. The investigation was based on a case study in which a maintenance system for an AT&T network was tested. For the purpose of comparison, two different models, both of which contained parameters and parameter partitions, were developed from the same specification. The models contained 72 and 162 combinations, C. For each i, $1 \leq i \leq C$, 10 abstract test suites were generated for each of the two models by random sampling without replacement. For every test suite, whether t-wise coverage with $t = 1, 2, .., n$ was calculated. Further, ten real test suite instances of each abstract test suite were created by randomly selecting a value for each parameter partition in each test case. The resulting test cases were then executed and the code coverage achieved by the complete test suite was monitored. Two code coverage measures were used, block (statement) coverage and path coverage.

The conclusion of this study was that a test suite that satisfies t-wise coverage, where $t \geq 2$, obtains comparable results in terms of block (statement) coverage. Thus, pair-wise coverage has comparable effectiveness as all-combination coverage with respect to block (statement) coverage. Larger values of t are required to achieve path coverage comparable to all-combination coverage.

Assume a test problem with three parameters a, b, and c, each with 2 possible values.

Let $tc1 = (a1,\ b1,\ c1)$, $tc2 = (a2,\ b2,\ c1)$, and $tc3 = (a1,\ b1,\ c2)$

The following nine pairs are covered by the three test cases:
$(a1, b1),\ (a1, c1),\ (b1, c1),\ (a2, b2),\ (a2, c1),\ (b2, c1),\ (a1, b1),\ (a1, c2),\ (b1, c2)$
Note that the pair $(a1, b1)$ occurs twice.

The following four pairs are not covered by the three test cases:
$(a1, b2),\ (a2, b1),\ (a2, c2),\ (b2, c2)$

$$2 - coverage = 8/12 = 75\%$$
$$2 - diversity = 8/9 = 89\%$$

Figure 14: An example of 2-coverage and 2-diversity of a test suite

B.1.3 Dalal and Mallows

Dalal and Mallows [DM98] provide a theoretical view of combination strategies. They provide a model for software faults in which faults are classified according to how many parameters (factors) need distinct values to cause the fault to result in a failure. A t-factor fault is triggered whenever the values of some t parameters are involved in triggering a failure. Each possible fault can be specified by giving the combination(s) of values relevant for triggering that fault. Further they provide two measures of efficiency of a combination strategy test suite. The t-wise coverage (in Dalal's and Mallow's words *t-coverage*) is the ratio of distinct t-parameter combinations covered to the total number of possible t-parameter combinations.

The *t-diversity* is the ratio of distinct t-parameter combinations covered to the total number of t-parameter combinations in the test suite. Thus, this metric summarizes to what degree the test suite avoids replication. Figure 14 shows an example of these two metrics.

B.1.4 Kuhn and Reilly

Kuhn and Reilly [KR02] have investigated 365 error reports from two large open source software projects, 194 from the Mozilla web browser and 174 from the Apache web server. Their aim was to find answers to the following three questions: (1) Is there a point in which a test suite that satisfies t-wise coverage is nearly as effective as a test suite that satisfies $t + 1$ coverage? (2) What is the appropriate value for t for particular classes of software? (3) Does this value differ for different types of software and by how much? Their findings indicate that a test suite that satisfies 4-wise

coverage would have found more than 95% of the errors. Further their results show that test suites that satisfies $2-6$ -wise coverage have similar effectiveness for both studied applications.

B.1.5 Grindal, Lindström, Offutt and Andler

Grindal, Lindström, Offutt, Andler [GLOA03] present the results of a comparative evaluation of five combination strategies. One of the combination strategies investigated, Each Choice, satisfies the each-used criterion. Two of the investigated combination strategies, Orthogonal Arrays and Heuristic Pair-Wise (AETG), satisfy the pair-wise criterion. The fourth, All Values, generates all possible combinations of the values of the input parameters. The fifth, Base Choice, satisfies the each-used criterion but also uses some semantic information to construct the test cases.

Except for All Values, which is only used as a reference point with respect to the number of test cases, the combination strategies were evaluated and compared with respect to the number of test cases, number of faults found, test suite failure density, and decision coverage achieved in an experiment that used five Unix command-like programs seeded with 131 faults.

As expected, Each Choice finds the smallest amount of faults among the evaluated combination strategies. Base Choice performs as well, in terms of detecting faults, as the pair-wise combination strategies despite fewer test cases. Their analysis of the results showed that the performance of Each Choice is unpredictable in terms of which faults will be detected. Moreover, Base Choice and the pair-wise strategies target different types of faults to some extent. The conclusion of the experiment was that Base Choice should be used alone for less important software and together with Heuristic Pair-Wise in a compound combination strategy for more critical software.

B.2 EP, BVA, CP and All Combinations

The three papers in this group considered aspects of choosing "interesting values" by techniques such as equivalence partitioning (EP), boundary value analysis (BVA), category partitioning (CP), and choosing all combinations of values.

B.2.1 Toczki, Kocsis, Gyimóthy, Dányi and Kókai

Toczki, Kocsis, Gyimóthy, Dányi, and Kókai [TKG+90] described the state of SYS/3, a software development tool. SYS/3 contains work processes and tools for several activities during systems development, including project management and control, document handling, and testing. A test case generator called T-GEN has been implemented for the Category Partition method (CP) [OB88]. In addition to handling the input space, T-GEN may also be used to generate test cases based on the output space in a similar manner as the input space is handled in CP. Also, T-GEN can generate complete or partial test scripts under certain conditions.

B.2.2 Kropp, Koopman and Siewiorek

Kropp, Koopman, and Siewiorek [KKS98] describe the Ballista testing methodology, which supports automated robustness testing of off-the-shelf software components. Robustness was defined as the degree to which a software component functions correctly in the presence of exceptional inputs or stressful conditions. The underlying testing strategy is an object-oriented approach based on parameter types and values instead of component functionality. The application of the Ballista methodology is demonstrated on several implementations of the POSIX operating system C language API, which consists of 233 calls. Together the POSIX calls use 190 test values across 20 data types. The expected results of the different tests are limited to "does not crash", "halts", and "returns a value". The simplest Ballista operating mode generates a test suite that has all combinations of all parameter values. If the total number of combinations is too large to be feasible, a deterministic pseudo-random sample is used. In the experiment reported in the paper, the limit for generating all combinations was set to 5000 test cases. Only seven of the POSIX functions tested require more than 5000 test cases.

B.2.3 Daley, Hoffman and Strooper

Daley, Hoffman, and Strooper [DHS02] showed how combination strategies can be used to test Java classes. A main focus of their work was on automation of test generation and execution. This is accomplished with the "Roast" framework, which consists of four steps. (1) *Generate* creates the combinations of parameter values to be tested, that is, the input portions of each test case. (2) *Filter* removes invalid combinations from the test suite. (3) *Execute* drives the execution of the test cases. (4) *Check* compares the expected with the actual output. Apart from a parameter model of the test problem, the expected output is the only manual intervention.

The combination generation includes two general strategies, both based on boundary value analysis. The tester provides information about the parameters, and for each parameter the different domains, or equivalence classes. The first strategy, k-boundary, is the Cartesian product of the boundaries of the different parameters. The second strategy, k-perimeter, is the boundary plus "the points in between." To illustrate the two strategies, assume a two parameter problem p_1 and p_2, with the integer domains [0, 1, 2, 3, 4] and [5, 6, 7, 8, 9]. The 1-boundary test suite contains the four test cases $[(0,5),\ (0,9),\ (4,5),\ (4,9)]$. The 2-boundary test suite uses the values closest to the extreme points resulting in the test suite $[(1,6),\ (1,8),\ (3,6),\ (3,8)]$. The 1-perimeter test suite contains the following 16 test cases: $[(0,5),\ (0,6),\ (0,7),\ (0,8),\ (0,9),\ (4,5),\ (4,6),\ (4,7),\ (4,8),\ (4,9),\ (1,5),\ (2,5),\ (3,5),\ (1,9),\ (2,9),\ (3,9),]$. These 16 tests form the complete perimeter.

k-perimeter significantly reduces the number of test cases for test problems with few parameters and many values per parameter, whereas k-boundary is more efficient when the opposite occurs, when there are many parameters with few values. In cases where parameter values depend on each other, Roast contains facilities to represent one large test problem with dependencies as several smaller test problems without dependencies.

B.3 Orthogonal and Covering Arrays

The six papers in this group considered aspects of choosing combinations by organizing the values into arrays.

B.3.1 Mandl

Mandl [Man85] may have been the first researcher who attempted to solve a testing problem with combination strategies. His focus was on orthogonal arrays. He states that the size of a test suite generated by OA is v^2. Underlying this statement is the assumption that all parameters contain exactly v interesting values. Mandl reports that OA was intended to design test cases used by the Ada Compiler Validation Capability (ACVC) test suite. No results from testing were reported.

B.3.2 Brownlie, Prowse and Phadke

Brownlie, Prowse, and Phadke [BPP92] reported results from a case study. The Orthogonal Arrays Testing System (OATS) tool, which is based on the OA combination strategy, was used to test a PMX/StarMAIL release. OATS automatically identifies orthogonal arrays and generates test suites that are based on descriptions of the parameters and interesting values of the test object.

In the case study OATS was applied both to the PMX/StarMAIL hardware and software configurations to the functionality of the system. 354 test cases were generated by OATS and an additional 68 test cases were created by hand to cover special conditions and to add tests that applied to specific configurations.

A previous release of PMX/StarMAIL was tested "conventionally." The results from this test were used as points of comparison when evaluating the results obtained by OATS and the manually created test cases. The authors estimated that 22 percent more faults would have been found if OATS had been used despite the fact that conventional testing have required twice as much time to perform. Further, they report that 1.5 years after the tested system was put into production, an additional three faults have been found that were missed in the case study. All of these faults were outside the scope of the planned testing.

The conclusion drawn by the authors is that OA should be used whenever there are many independent hardware, software, or functional parameters to test.

B.3.3 Williams and Probert

Williams and Probert [WP96] illustrated the applicability of OA to test network elements of a telephony system. The OA combination strategy was used to select system configurations for testing rather than individual test cases. The authors pointed out that the cost for setting up a certain configuration is often substantial. Further, each configuration requires complete test suites to be executed, which adds to the cost of testing the configuration. For this reason, it is important to find ways to identify a small number of configurations to test. One of the main aims of the

paper was to present a self-contained guide to apply the theory of OA. Thus, most of the paper is spent on describing OA and the method to construct orthogonal Latin Squares of desired sizes and how to handle various practical situations that are not handled by the basic OA combination strategy.

B.3.4 Williams

Williams [Wil00] continues to explore the applicability of pair-wise coverage to configuration testing. In this paper, Williams introduced Covering Arrays (CA) as an alternative to Orthogonal Arrays. A comparative analysis of the number of test cases generated by CA and IPO is reported. CA and IPO behaved similarly, often to the advantage of CA, with only minor differences in number of test cases for the 16 different test problem sizes. However, CA outperforms IPO by almost three orders of magnitude for the largest test problems in terms of time taken to generate the test suites.

B.3.5 Williams and Probert

Williams and Probert [WP01] stated that the use of an interaction test coverage metric allows testers to both evaluate already existing test suites and to generate new test suites that satisfy a predetermined level of interaction. Based on this, Williams and Probert suggest the use of an *interaction element*, which may be used in the same way statements and decisions are used for code coverage. An interaction element is a specific (sub)combination of parameter values. By reasoning about types of interaction elements, Williams and Probert formally defined a set of coverage criteria, including t-wise coverage.

Williams and Probert showed that the general problem of finding a minimal set of test cases that satisfies t-wise coverage can be NP-complete.

B.3.6 Williams and Probert

Williams and Probert [WP02] continue the work from Williams previous paper [Wil00] in finding an algorithm that will generate a minimal test suite that satisfies pair-wise coverage. Previously, they provided a fast approximate algorithm by using different building blocks. This time they show how the interaction test coverage problem may be formulated as a $[0, 1]$ integer programming problem. Using general solution mechanisms for such problems, it is possible to find a minimal solution. These problems have previously been shown to be NP-complete, which indicates that finding an optimal solution for the interaction test coverage problem may also be NP-complete. The uncertainty comes from the fact that not all $[0, 1]$ integer programming problems can be converted to an interaction test coverage problem.

B.4 AETG

A test tool called Automatic Efficient Test Generator (AETG) was created at Bellcore (now Telcordia) and a number of papers were published that used it in practical industrial settings.

B.4.1 Burroughs, Jain and Erickson

Burroughs, Jain, and Erickson [BJE94] presented the first use of the Telcordia (Formerly Bellcore) test tool called Automatic Efficient Test Generator (AETG). They illustrated the use of AETG with two examples in which the test suite generated by AETG is compared with corresponding test suites generated by AC and PPW on the basis of number of test cases and pair-wise coverage (breadth of coverage in their terms). AETG produced similar results in terms of test cases as PPW and an order of magnitude lower number of test cases than AC. By definition AETG and AC reach 100% pair-wise coverage while PPW only reaches $30-35\%$. The authors concluded that AETG is an effective and efficient way to create test cases.

B.4.2 Cohen, Dalal, Kajla and Patton

Cohen, Dalal, Parelius and Patton [CDKP94] described how to use AETG in the context of screen testing, that is, testing the input fields for consistency and validity across a number of screens. From previous investigations they knew that most of the faults in the screens are caused by the value of one single parameter or the interaction of two parameters. Thus, although AETG can generate a test suite for t-wise coverage for an arbitrary t, the authors predicted that pair-wise coverage will be what testers will need.

The authors theoretically compared the underlying algorithm of AETG, without revealing it, with OA. They showed that by removing the requirement of balance in OA, AETG is able to reduce the number of test cases needed to for pair-wise coverage. Further, the authors pointed to several other advantages of AETG over OA, for example, that the algorithm is always able to find a solution to a test problem.

The authors describe the general features of AETG and conclude by reporting the results of a small experiment in which AETG was used to test ten screens of an inventory database system. No comparative analysis of the results was given.

B.4.3 Cohen, Dalal, Parelius, Fredman and Patton

Cohen, Dalal, Parelius and Patton [CDPP96] presented the first publicly available descriptions of the algorithms implemented in the Automatic Efficient Test Generator (AETG). The same information with more details was later presented by Cohen, Dalal, Fredman, Patton [CDFP97].

Results of two experiments were described. In the first experiment, AETG was used to test ten Unix commands and the code coverage was measured. The authors report that pair-wise tests resulted in over 90% block coverage. Specifically, Cohen et al. experimented with the Unix

command "sort." Different ways of modeling the parameters and values were tried and the results were compared with the results from a randomly generated test suite, and from "all" combinations. The AETG generated test suites consistently produced better coverage than the randomly produced test suites.

Further, Cohen et al. used the AETG system to test two versions of a Telcordia product. A total of 75 parameters were identified, and 28 test cases were created to satisfy pair-wise coverage. Even though the AETG tests were executed at the end of the normal product development, several new faults, both in the specification and in the implementation, were detected. The main conclusion drawn by the authors was that combination strategies in general and the AETG system in specific are both effective and efficient at functional testing.

B.4.4 Burr and Young

Burr and Young [BY98] report on using AETG to test Nortel software. They tested a third party email product that converts email messages from one format to another. The most important step according to the authors was the modeling of the input space, in their case the possible ways to compose email messages. An augmented BNF that describes the syntax and format of the e-mail messages was used. Thus the authors were faced with the challenge of converting these BNF rules into AETG constructs. The AETG model was iteratively constructed and the final version was used to create a suite of test vectors, i.e., test case inputs. The next step was to convert these abstract test vectors into real test cases. A Perl script was written that converted the AETG representation of the test case inputs to real email messages.

The code coverage of the tests was monitored for two purposes. First, they wanted to verify how complete the specification was. Second, they wanted to measure the code coverage of AETG tests. Initial experiments showed that ad hoc testing resulted in about 50% block and decision coverage. By continually applying code coverage as the AETG models were refined, decision coverage was increased to 84%. The power of AETG combined with automatic translation into real test cases made it easy to find test cases that contributed to the coverage. Usually, the code needs to be analyzed to find test cases that will increase coverage. Due to the simplicity of generating test suites, many "what if" scenarios could be tested without increasing the number of test cases. This turned the traditionally cumbersome analysis of the code into a trial-and-error type of process.

To compare the AETG results with an alternate test generation technique, a restricted version of BC was used to create a test suite from the same input model. Of the reachable code, AETG reached 93% block coverage with 47 test cases, compared to 85% block coverage for the restricted version of BC using 72 test cases.

B.4.5 Dalal, Jain, Karunanithi, Leaton and Lott

Dalal, Jain, Karunanithi, Leaton, and Lott [DJK+98] reported results from two experiments in which principles of model-based testing were used to automatically generate test cases for Bellcore's Intelligent Service Control Point. The authors reported two significant advantages over manual

testing. The first is coping with change. They found that changes to the product or the test criteria have less impact on the test organization if test cases can be regenerated. The second is that a systematic choice of test cases, in their research created by AETG, can increase the quality of the product by detecting more faults than with manual testing.

In addition to selecting the test cases, the work also included the implementation of a test scaffolding to automatically execute the test cases and compare actual and expected results. The implementation of the scaffolding was the most labor intensive part of this work.

In total, more than 5000 test cases were generated and executed, of which about 400 failed. The authors reported four lessons learned. First, the model of the test data is fundamental and requires considerable domain expertise. Second, model-based testing is in fact a development project since it consists of much testware that needs to be developed and maintained. Third, change must be handled and unnecessary manual intervention should be avoided. Fourth, technology transfer requires careful planning, that is, changing a work process or a state-of-practice is a project in itself.

B.4.6 Dalal, Jain, Karunanithi, Leaton, Lott, Patton and Horowitz

Dalal, Jain, Karunanithi, Leaton, Lott, Patton, and Horowitz [DJK+99] presented the architecture of a generic test-generation system based on combination strategies. A test data model is derived from the requirements and then used to generate test input tuples. These are in turn refined into real test inputs and expected outputs. The test inputs are fed to the system under test, which generates actual output. Finally the actual and expected outputs are compared.

The authors claim that the model-based testing depends on three key technologies: the notation used for the data model, the test generation algorithm, and the tools for generating the test execution infrastructure including expected output. The notation and algorithms can be used in multiple projects but not the tools.

The authors also present the results from four case studies in which their model-based test architecture has been used. Two of these case studies were reported elsewhere [DJK+98]. The third case study concerned testing a rule-based system used to assign work requests to technicians. Five separate models were used to generate tests for the system. Two types of tests were generated and run. The first type concerned the initial assignment of jobs at the start of the day. The second type of tests focused on the dynamic assignment of tests during the course of the day. The latter was much harder, since these tests depend on the current state of the database, which dynamically changes. This, in turn, means that checking actual output is very difficult since all data is related. In this case study the checking of the data was performed manually to a large extent. Thirteen tests were run, resulting in block coverage of 84%. Although relatively good, the authors suggest that block coverage may not be significant for an object-oriented system that implements a rule processing mechanism. The tests revealed four failures. The authors claim that the relatively small amount of faults found was because checking the results was complicated.

The fourth case study consisted of testing a single GUI window in a large application. Due to

the complexity of the window the creation of the input model turned out to be the most difficult task. 159 test cases were generated and executed automatically since the expected results were easy to describe and check. Six failures were revealed and reported, three of which were uniquely discovered by the automatic testing approach.

The main lessons learned were that the model of the test data is fundamental and iteration is often required to find a suitable model. The authors also saw benefit in designing the test scaffolding with intermediate results stored in files, which can be manually changed. When executing thousands of tests automatically, the tester must be able to start the execution at an arbitrary point in the test suite without having to worry about the state of the test object. Finally, the authors give advice on how to change a process and how to introduce new tools.

B.4.7 Dalal, Jain, Patton, Rathi and Seymore

Dalal, Jain, Patton, Rathi, and Seymour [DJP+99] described a publicly available web interface that allows testers to use AETG. The tester inputs the functional model of the test object either through a set of web GUI widgets or through a text-based notation. AETG can produce up to three test suites. First is a valid test suite, which contains valid combinations of parameter values. Second is an invalid test suite, which contains test cases that violate constraints in parameter value combinations, as defined by the user. Third is an illegal test suite, which contains test cases with values explicitly declared to be illegal by the tester.

B.5 Base Choice

One paper was published that introduced the idea of base choice.

B.5.1 Ammann and Offutt

The seminal work on BC by Ammann and Offutt [AO94] theoretically compared the number of test cases needed to fulfill base-choice coverage with the number of test cases needed to fulfill each-used coverage and N-wise coverage. They concluded that BC is relatively inexpensive and yet effective.

Ammann and Offutt also demonstrated the feasibility of BC with a small experiment. MiStix is a simple file handling system with ten operations for creating, deleting, copying, and moving files and directories. The implementation is about 900 lines of C code. A complete Z specification had been developed for MiStix for a previous study. A complete BC test suite was generated from the Z specification by hand. The paper reported extensive information about the identification of parameters and interesting values of each parameter. In particular, the paper illustrated how Z preconditions and postconditions can be used to identify parameters and interesting values more or less mechanically.

The final BC test suite contained 72 test cases for the ten operations. Five of the 72 test cases were duplicates and some of the test cases were superseded by others in the sense that they were preconditions of other test cases. The test suite detected 7 out of 10 known faults.

B.6 In Parameter Order

Two papers appeared on the subject of choosing values based on the order of parameters.

B.6.1 Lei and Tai

Lei and Tai describes an algorithm called In-Parameter-Order (IPO) for generating a test suite that satisfies pair-wise coverage [LT98]. An extended version of the same paper was presented later [LT01] and an extract was also published [TL02]. The papers describe the framework of the IPO strategy, an optimal algorithm for the vertical growth of the test suite, and two alternative algorithms for the horizontal growth of the test suite. The algorithms have been implemented in Java in a tool called PairTest, which provide a graphical user interface to make the tool easy to use.

The IPO combination strategy is compared with the AETG combination strategy in the paper. The two combination strategies are shown to perform similarly with respect to number of test cases for six different test problem sizes. Tai and Lei also show that the time complexity of IPO is superior to the time complexity of AETG. IPO has a time complexity of $O(v^3N^2log(N))$ and AETG has a time complexity of $O(v^4N^2log(N))$, where N is the number of parameters, each of which has v values.

B.6.2 Huller

Based on a real example of testing ground systems for satellite communications, Huller [Hul00] showed that pair-wise configuration testing may save more than 60% in both cost and time compared to quasi-exhaustive testing. The set of configurations that satisfied pair-wise coverage was developed by hand via an algorithm that resembles IPO, although no reference is given.

B.7 Antirandom Testing

An approach for choosing values based on the opposite of random selection has been discussed in two papers.

B.7.1 Malaiya

Malaiya [Mal95] presented seminal work on antirandom testing. Tests are added in sequence and each value is represented as a sequence of bits. Antirandom means that each new test case added

resides as far from all previous test cases in the sequence as possible, using Hamming or Cartesian distance measured on the bit representation.

The paper presented two algorithms to create an antirandom sequence of test cases. The first is a naive method that requires an exhaustive search. The second exploits properties of an antirandom sequence of test cases to create the final result more quickly. Further, the paper illustrates how checkpoint encoding can be used to identify a mapping between the parameters of the test problem and the corresponding test cases. A number of considerations and observations for this mapping are listed.

For the selected checkpoints, the intent of antirandom testing is to generate all possible combinations. By executing the test cases in the sequence they are generated, that is, as far apart from each other as possible, the hypothesis is that antirandom testing is likely to detect the presence of faults earlier than random testing. No data to support this hypothesis was presented in the paper.

B.7.2 Yin, Lebne-Dengel and Malayia

Yin, Lebne-Dengel, and Malayia [YLDM97] used three benchmark programs to compare antirandom testing with random testing. Test cases were generated one by one according to the different evaluated combination strategies. Each new test case was executed and the increase in branch, loop, relational, and total coverage was monitored. The antirandom combination strategy consistently performed well by reaching high coverage with few test cases. The authors concluded that the results suggest that antirandom testing might be a worthwhile alternative to other black-box test methods. However, the investigated benchmarks are small and might not be representative. Also, the encoding used for the tests can have a strong affect on the results, so encoding rules need to be further investigated.

B.8 Fractional Factorial Designs

Selecting test values has a strong relationship to scientific experiments, and the experimental method of factorial design was suggested to help select values.

B.8.1 Heller

Heller [Hel95] uses a realistic example to show that testing all combinations of the parameter values is infeasible in practice. The paper concluded that there we need to identify a subset of combinations of manageable size. Heller suggests using fractional factorial designs. These designs have traditionally been used in physical experiments to decrease the number of experiments. The underlying mathematics of the fractional factorial designs allows researchers to establish which factors cause which results. Heller does not describe specifically how is use fractional factorial designs in software testing. The paper explains how to translate a test problem into a representation for factorial designs. It is also shown that the number of test cases from fractional factorial designs is less than all combinations.

B.8.2 Berling and Runesson

Berling and Runesson [BR03] describe how to use fractional factorial designs used for performance testing. Specifically, they focus on how to determine the effect of different parameters on the performance of a system with a minimal or near minimal set of test cases. Berling and Runesson state that when the result variable has a nominal type, (for example, true or false) the property of fractional factorial designs cannot be exploited.

B.9 Missing Papers

The following two papers have proven hard to acquire. According to the papers referencing these papers, both of them contain information about attempts to automate the generation of test cases according to some combination strategy algorithm. The papers are included in this survey for completeness.

B.9.1 Biyani and Santhanam

This paper by Biyani and Santhanam [BS97] contains information about a tool called TOFU. It is referenced by Kropp et al. [KKS98].

B.9.2 Sherwood

This paper by Sherwood [She94] contains among other things a description of a tool called CATS implementing an early heuristic algorithm for pair-wise coverage. The paper is referenced by Dunietz et al. [DES+97] and by Cohen et al. [CDPP96, CDFP97, DM98].