A Novel Test Case Generation Method through Metamorphic Priority for 2-Way Testing Method UMBCA Implementation Criteria

Er. Sumit Jain¹, Er. Mohsin Sheikh²
¹Acropolis Technical Campus, Indore Bypass Road, Near Tillore Village, Rala Mandal, Indore, M.P, INDIA
²Medi-caps Group of Institutions, A.B. Road, Pidgamber Rau, Indore, M.P, INDIA

ABSTRACT
Increased size & complexity of software needs better approaches for different functionalities in the software development life cycle. Quality assurance of software is mainly done by ways of testing, an activity that faces constraints of both time & resources. So combinatorial testing is a well known dynamic approach for quality improvements because it provides effective error detection at very low cost. Generating an optimal set of test will effectively test the software system by pairing the input parameters through pairwise testing using orthogonal arrays. Hence an efficient strategy is required to reduce the number of test cases formed by above mentioned method.

In this paper we propose a new approach based on UML modelling which apply the test prioritization activity. For this we use metamorphism operators & apply it on our unified modelling based combinatorial architecture (UMBCA). Primary result obtained on a test data indicates that the techniques are effective & easy. The paper makes frequent reference to PICT, an existing & publicly available tool built on top of a flexible combinatorial test case generation engine.

Keywords- Pairwise Testing; Test Suite Prioritization; UML Mode-ling; UMBCA Architecture; Metamorphism Operator; PICT Tool.

I. INTRODUCTION
Testing process consists of designing a set of good test cases. It takes set of inputs, executes programs with test cases and examines the results produced after execution. Test cases can be mapped directly to, and derived from use cases. Test cases can also be derived from system requirements. One of the advantages of producing test cases from specifications and design is that they can be created earlier in the development life cycle and be ready for use before the programs are constructed. Additionally, when the test cases are generated early, software engineers can often find inconsistencies and ambiguities in the requirements specification and design documents.

Combinatorial testing (ct) is a black box system testing technique that samples inputs, configurations and parameters and combines them in a systematic way. In our strategy we are specifically focusing on pairwise testing (type of combinatorial testing) for test prioritization for better performance. Pairwise testing is an effective test case generation technique that is based on the observation that most faults are caused by interactions of at most two factors [1]. The technique has been known for almost 20 years, but it is only in the last five years that we have seen a tremendous increase in its popularity. Pairwise-generated test suites cover all combinations of two therefore are much smaller than exhaustive ones yet still very effective in finding defects. In our approach we strongly focus on genetic algorithm methodology. It works very well on mixed (continuous and discrete), combinatorial problems. They are less susceptible to getting 'stuck' at local optima than gradient search methods. But they tend to be computationally expensive [2].

The need for prioritization arises in many different phases of the software testing. For instance, requirements need to be prioritized; tasks on schedules need to prioritize; and testers may have to prioritize the tests that they run. A prioritized interaction test suite may be particularly useful to testers who want to test key areas of concern first, or for those that wish to run their tests in order of importance until they exhaust their available resources[3]. For instance, a tester may request a complete test suite and attempt to run as many tests as they can budget. In this context, it is important to test the most important items first.

Using our methodology, test cases can be prioritizing in an ordered based on maximum coverage of pairs. We propose a new metamorph based priority by applying it to test case with genetic algorithms. Our proposed methodology is centered on uml (umba approach) which is a widely accepted set of notations for modeling object-oriented system [4, 5]. It has various diagrams for depicting the dynamic behavior of objects in a system. Our work presents an idea of creating test cases that are automatically generated from the requirement
specification based on UML.

II. BACKGROUND

(a) The purpose of the study

This study aims to improve an automated test case generation method to minimize a number of test cases while maximizing an ability to identify critical domain specific requirements. It has been proven that the software testing phase is one of the most critical and important phases in the software development life cycle. In general, the software testing phase takes around 40-70% of the effort, time and cost [6]. This area has been well researched over a long period of time. Unfortunately, while many researchers have found methods of reducing time and cost during the testing process, there are still a number of important related issues that need to be researched. This study introduces a new test case generation process with a requirement prioritization method to resolve the following research problems:

1. Inefficient test case generation techniques with limited resources.
2. Lack of an ability to identify critical domain requirements in the test case generation process.
3. Inefficient automated test case generation techniques and,
4. Ignoring a number of generated test cases.

(b) Using Model Based Testing

Unified Modeling Language (UML) is a standardized general-purpose modeling language in the field of object-oriented software engineering. UML combines techniques from data modeling (entity relationship diagrams), business modeling (work flows), object modeling, and component modeling [7]. It can be used with all processes, throughout the software development life cycle, and across different implementation technologies. So by combining UML with pairwise testing has become an indispensable tool in a software tester’s toolbox (UMBCA Tool).

Most tools, however, lack practical features that are necessary for them to be used in the industry.

In this paper we propose a new method based on metamorphism operator of genetic algorithm for assigning priority to test suites so as to provide complete coverage. In particular, it does focus on ways in which the pure pairwise-testing approach must be modified to become practically applicable, and on the features that UMBCA tools must offer to support the tester who is trying to use pairwise testing in practice [8].

This paper makes frequent references to PICT, an existing and publicly available tool for comparison of our tool results that is built on top of a flexible combinatorial test-case–generation engine, which implements several of the concepts that are described here in [9, 10].

A set of possible inputs for any nontrivial piece of software is too large to be tested exhaustively. Techniques such as equivalence partitioning and boundary-value analysis help convert even a large number of test levels into a much smaller set with comparable defect-detection power [11, 12]. Still, if software under test (SUT) can be influenced by a number of such factors, exhaustive testing again becomes impractical.

In summary the study aims to:

1. Prioritize a huge set of requirements in order to improve the effectiveness of test case generation techniques while there are limited resources,
2. Increase the ability to cover more critical domain requirements during the generation process and
3. Minimize and generate a small set of test cases with high ability to reveal faults.

III. PROPOSED METHOD

(A) Proposed Method Structure:

A Pairwise software testing is been always used to provide maximum coverage with minimum efforts. I am using pairwise testing with requirement prioritization so as to create test suite with the help of UMBCA design. as follows:

Design test cases: The purpose of this step is to generate and prepare a set of test cases. Therefore, the outcome of this step is a set of test cases. A set of test cases may be represented in Excel format, as Word documents or as a database. We are assuming a test case with 4 parameters & having 11 values.

Prepare test data: The purpose of this step is to generate and prepare test data for each test case. The outcome of this step is a set of test data. By test data we mean the values of a parameter is assigned a specific index for there position. We calculate the initial position & find out the number of pairs used or unused in data set.

Run program with test data: This is an execution test step. Test case and test data will be run in this step. The result of this step is actual system output. In this step a exhaustive searching is perform to identify the test set & arrange them in a priority based on usage.

Compare results to test cases: This step is used to compare the system output to expected output in the test case. The milestone of this step is a test report of running the test case with test data. We use PICT tool to compare the result obtain by our UMBCA tool [13].

The review shows that existing test case generation techniques derive test cases directly from requirements, specification requirement documents or diagrams. None of them are concerned with prioritizing a huge set of requirements. Practically, there are a huge set of requirements in the software development, particularly in large complex software. Thus, prioritizing requirements before preparing and generating test cases is one of the most important activities, which software test engineers can not ignore.

This study proposes to insert an additional process in
order to maximize critical domain specific requirements while minimizing a number of test cases during testing process. The requirement prioritization process contains primarily two major processes: (1) requirement classification by first two phases of UMBCA method (Setting us test criteria & Test model design) and (2) requirement prioritization by last three phases (Test suite creation, Performing test, & Result Analysis).

(B) UMBCA Support Algorithm

The essence of how UMBCA works is to generate one test set at a time, using genetics & greedy algorithms to place each parameter value, until all possible pairs have been captured. The high-level algorithm provides the parameter interaction with multidimensional arrays. It will dynamically reduces the complexity of test identification. Our UMBCA architecture is well clarify & easy to apply. This algorithm will shows how implementation occurs with prioritizing requirements.

Selecting the best candidate with Test Model: The model represents the expected behavior of the System under Test in UMBCA method. Here Unified Modeling Language (UML) is used to formalize the control points and observation points of the system, the expected dynamic behavior of the system [15]. The model is been selected on the basis of operator we are using. For genetic algorithm in combination of greedy approach we uses metamorphism operator. We can also select crossover operator.

Adding the best candidate to test Sets (Test Suite Creation): is an automated process that generates the required number of abstract test cases from the test model. Each generated abstract test case is typically a sequence of high-level SUT actions, with input parameters and expected output values for each action of the test repository is done by updating the test model, then automatically regenerating the test suites.

Updating data structures operations: Generated concrete tests are typically executed within a standard automated test execution environment of UMBCA implemented. Alternatively, it is possible to execute tests manually – i.e. a tester runs each generated test on the SUT, records the test execution results, and compares them against the generated expected outputs [16].

Analyze Test Result: Analyze the real system which is to be tested and accepted by user. The effectiveness of test cases can be evaluated using a fault injection technique called mutation analysis. Mutation testing is a process by which faults are injected into the system to verify the efficiency of the test cases. For this we are using pairwise approach whose problem domain is NP Complete so the solution must be in accordance with UMBCA [17]. Referring PICT tool for result comparison of generated output by our proposed method will sometimes create problem when considering seed values of parameter, so it may produce different result with different seed values.

(C) Use of Genetic Algorithm

The problem of generating a minimum test suites is NP Complete. In this section we show that how a pairwise coverage problem is been represented by deterministic decision making through NP complete and proposed a way to solve this through genetic algorithms using its SMV and mutation strategies (Crossover, Mutation are commonly known as metamorphs because of having more than one forms) [2]. A strategy for using Genetic Algorithms (GAs) to solve NP-complete problems is presented [4, 18]. The key aspect of the approach taken is to exploit the observation that, although all NP-complete problems are equally difficult in a general computational sense, some have much better GA representations than others, leading to much more successful use of GAs on some NP-complete problems than on others. Since any NP-complete problem can be mapped into any other one in polynomial time, the strategy described here consists of identifying a canonical NP complete problem on which GAs work well, and solving other NP-complete problems

![Fig: A UMBCA based design implementation algorithm for Pairwise testing](image)

The above algorithm can be separated in five parts

**Generating Test Criteria on the basis of Requirement specification:** Candidates can be generated by generation criteria which are based on structural model coverage, using famous test design strategy of pair-wise testing. Another useful kind of test generation criteria ensures that the generated test cases cover all the requirements, perhaps with more tests for requirements that have a higher level of risk so in this paper we are combining the pairwise approach with new architecture through UML Navigational approach [14].
indirectly by mapping them onto the canonical problem [19].

To use a genetic algorithm, you must represent a solution to your problem as a genome (or chromosome). The genetic algorithm then creates a population of solutions and applies genetic operators such as mutation and crossover to evolve the solutions in order to find the best one(s). You can use any representation for the individual genomes in the genetic algorithm. Related work primarily with strings of bits, but you can use arrays, trees, lists, or any other object. But you must define genetic operators (initialization, mutation, crossover, comparison) for any representation that you decide to use [2].

Remember that each individual must represent a complete solution to the problem you are trying to optimize. The mutation operator introduces a certain amount of randomness to the search. It can help the search find solutions that SMV alone might not encounter. A formal specification is a repository of knowledge about a system, and a recent method uses such specifications to automatically generate complete test suites via mutation analysis [20]. We define an extensive set of mutation operators for use with this method. We report the results of our theoretical and experimental investigation of the relationships between the classes of faults detected by the various operators. Finally, we recommend sets of mutation operators which yield good test coverage at a reduced cost compared to using all proposed operators specification, producing mutant specifications. Better test sets are those which reveal more mutants.

**IV. EMPIRICAL RESULT**

Result Calculation: Pairwise testing is a technique that allows you to reduce a large, unmanageable set of test-case inputs to a much smaller set that is likely to reveal bugs in the system under test. So for result analysis of our UMBCA approach a tool is implemented based on genetic transmutation & crossover operator combining called as metamorphs. We always compare our tool structure & results with PICT tool which is free to use publicly available tool for pairwise testing. For this we consider the dummy Windows Form-based application. The application has four input parameters with different values.

Parameter 1: a, b
Parameter 2: c, d, e, f
Parameter 3: g, h, i
Parameter 4: j, k

So one test-case input set would be \{“a”, “c”, “g”, “j”\}. The dummy application has a total of \(2 * 4 * 3 * 2 = 48\) possible input sets, which is certainly manageable. But imagine a card-playing application of some sort with five parameters, where each parameter can take on one of 52 values (to represent a card from a normal deck of playing cards, with replacement). In this situation there would be \(52 * 52 * 52 * 52 * 52 = 380,204,032\) possible input sets, which is likely to be unmanageable unless you could programmatically generate expected values for each test set input.

The idea of pairwise testing is to generate a list of test sets that capture all possible pairs of parameter values from each parameter. For the above example shown, there are a total of 44 such input pairs: \((a,c), (a,d), (a,e), (a,f), (a,g), (a,h), (a,i), (a,j), (a,k), (b,c), (b,d), (b,e), (b,f), (b,g), (b,h), (b,i), (b,j), (b,k), (c,g), (c,h), (c,i), (c,j), (c,k), (c,l), (d,g), (d,h), (d,i), (d,j), (d,k), (e,g), (e,h), (e,i), (e,j), (e,k), (f,g), (f,h), (f,i), (f,j), (f,k), (g,j), (g,k), (h,j), (h,k), (i,j), (i,k)

Now the test set \{“a”, “c”, “g”, “j”\} captures six of the 44 pairs: \((a,c), (a,g), (a,j), (c,g), (c,j)\ and \(g,j)\). So the goal of pairwise test set generation is to produce a collection of test sets that capture all 44 pairs. So our tool is generating a collection of 12 test sets that capture all 44 input pairs for the scenario.

Result Test Prioritize test sets:-
If you trace through each pair of values in the 12 test sets generated, you’ll see that they do in fact capture all 44 pairs listed above. So in this situation, we have reduced our possible test-case inputs from 48 test cases to 12 test cases. The savings aren’t very significant for this small example, but as I’ll show in a moment, using pairwise testing can dramatically reduce the number of test-case inputs in many situations. The underlying assumption of pairwise testing is that software bugs are more frequently found in code that involves the interaction of values from different parameters than in code that involves values from within a particular parameter & can be visualize by UML methodology.[21,22].

**Empirical Study:** We use our theoretical technique to analyse existing testing methods. In particular, we show that the basic meaningful impact strategy is stronger in that it tests for mutations from a harder-to-detect mutation operator than was claimed by the authors. We also and that if we detect mutants generated by the insertion mutation operator, we get almost perfect pairwise coverage.

<table>
<thead>
<tr>
<th>Testing Tool/Type</th>
<th>4 Parameter (Total 11 Values)</th>
<th>20 parameter (10 Values Each)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Without Pairwise Testing</td>
<td>48 Test Suites</td>
<td>10^20 Test Suites</td>
</tr>
<tr>
<td>PICT Tool</td>
<td>14 Test Suites</td>
<td>217 Test Suites</td>
</tr>
<tr>
<td>UMBCA Tool</td>
<td>12 Test Suites</td>
<td>216 Test Suites</td>
</tr>
</tbody>
</table>

**V. EXPECTED BENEFITS**

Previous algorithms used in tools to generate software interaction test suites have been evaluated on criteria of accuracy, execution time, consistency, and adaptability to seeding and constraints. This paper discussed a further important criterion: prioritization based on user specified importance. An algorithm was described & computational results suggest that the method provides a useful, and simple, mechanism for generating prioritized test suites. Greedy methods for constructing biased covering arrays can be useful when testers desire a prioritized ordering of tests. Through the implementation of our test cases, faults can be rapidly detected assisting in the development of feature model analysis tools and improving their reliability and quality. For its design and evaluation, we used popular techniques from the software testing community to assist us on the creation of a representative set of input-output combinations. These test cases can be used either in isolation or as a suitable complement for further testing methods such as white–box testing techniques or automated test data generators. As suggested by the testing literature, each test case was designed to reveal a single type of fault. This allows users to identify clearly the source of a fault once it has been detected.

**VI. CONCLUSION**

Pairwise testing is a combinatorial technique with probabilistic factors. Pairwise test set generation is an important technique, but it isn’t magic. Remember that pairwise techniques simply reduce the number of test-case inputs in situations where you just have too many test cases to deal with. Pairwise test set generation does not create test-case expected results. You should always begin by using normal testing principles, such as looking at boundary conditions, using pure random input and so on, and then use pairwise testing to supplement your test-case generation. Additionally, as a general rule of thumb, more testing is better, so there’s no reason why you can’t add additional test-case inputs to those produced by pairwise generation tools. Although pairwise testing is useful in many situations, be sure to use it only when appropriate.

I have found pairwise test set generation to be very useful for configuration testing, for module testing methods that accept enumerated values and for testing SQL databases where each column in a table has a relatively small number of different values. Pairwise testing is not necessarily a good approach for scenarios where you have a relatively small number of test-case inputs, or when you can programmatically produce test-case expected results (and therefore deal with a large test-case input set). And pairwise testing is not normally usable when the input values to the system under test are not discrete. However, even in situations where the number of possible parameter values is very large, you may be able to effectively use pairwise test-case input generation by separating parameter values into equivalence classes. When used properly, pairwise test set generation is an important technique that can help you produce better software systems.
The empirical study & the above comparison graph easily prove that the strategy which we are proposing is effective & efficient for pairwise coverage problem. The test cases will 70% reduced in most of the cases depending upon the seed value selected as per the given test criteria based on requirement. Future work will more effectively enhance the above proven results in a systematic way so as to generalize the tool. Also some researchers was focusing on improving solution domain of this genetic theory by NP complete & hard relations. So while proposing a new strategy in combination with pairwise approach we always kept in mind its practical implementation so as to make the tester’s work easy.UMBCA Tool is a great deal in test suite reduction & in addition it also provided maximum coverage based on genetic theory.

REFERENCES