Metamorphic testing is a new and innovative technique, which is used to determine if a test execution uncovers a fault. This is a more practical method than the test oracle method. Oracle is the big issue in testing that only compares the generated output to the predicted output. It used as a component for figuring out results either successful or not. By contrast, metamorphic testing applies a modification to a test input and utilizes metamorphic relations. It then observes how the program output changes into a different one as a result metamorphic testing change the way that software is tested the faults. This paper illustrate about the overview about metamorphic testing, various benefits of metamorphic testing technique with contrast to another techniques of software testing and various challenges regarding metamorphic testing.

1. INTRODUCTION
This state of the art technique is relatively novel. Metamorphic testing was first introduced in 1998[1]. In the past 18 years, this technique has been applied in an array of fields such as web services and computer graphics. The goal of this report is discuss and summarize research on metamorphic testing. Usually, software testing consists of running a certain program using test inputs and then checking the results. In order to check these results, a test oracle is used. This oracle usually consists of checking the output by comparing an expected result with the observed output. This process can be very costly at times. Other times, an oracle does not exist. There is also substantial room for error. This probability is particularly considerable when dealing with programs which produce complex output, for
example, complicated numerical simulations or a code generated by a compiler. In these cases predicting the correct output and then comparing it to the expected output will be likely to cause error. This particular problem is known widely in the software testing community as the oracle problem and it is one of the biggest challenges of software testing.

2. HISTORY OF METAMORPHIC TESTING

The very first hints of the concept of metamorphic testing can be found in some works which were written before 1998. The first introduction of metamorphic testing, as it is known today, is found in a technical report by Chen et al. The subject of metamorphic is a promising field which will bring breakthroughs in research. Today the use of software around the world has been growing exponentially. At the same time there have been increasing reports of faults within software. Both the software industry as well as the academic community has in interest in the assurance of quality software. This assurance of quality software is done through software testing. The goal of software testing is to detect as many faults in software possible and to detect these faults in the quickest method possible. An oracle can be used to verify the correctness of a test result for any given test case. As mentioned before, an oracle may be too expensive or simply not available. That is why metamorphic testing is a more practical option. While, there have been other ways of solving the oracle problem [2].

3. TECHNIQUES FOR FINDING FAULTS

I. One method is the construction of oracles from formal specifications. For example Heiron developed algorithms for two types of conformance relationships. This was to test physically distributed systems using the framework of finite state machines. One must note that this approach requires specifications which are expressed in a formal notation form. However this is not always possible. For this reason metamorphic testing is more practical. Metamorphic testing is applicable regardless of how the specifications may be written [2].

II. Another technique which has been used is called the assertion checking technique. During the programming phase, assertions are embedded in the source code. These assertions constrain different portions of the source code. When executing the program, one can find faults in the software by looking for violations on the constraints at runtime. In contrast, metamorphic testing requires that the program be executed multiple times. However, when comparing both these methods, it has been demonstrated that metamorphic testing consistently finds more fault than assertion testing. Metamorphic testing therefore has a greater fault detecting capability. A downside is though, that metamorphic testing usually incurs additional overheads than incurred under the assertion checking technique.
In contrast, metamorphic testing requires that the program be executed multiple times. However, when comparing both these methods, it has been demonstrated that metamorphic testing consistently finds more fault than assertion testing. Metamorphic testing therefore has a greater fault detecting capability. A downside is though, that metamorphic testing usually incurs additional overheads than incurred under the assertion checking technique. An additional technique which is used the technique of utilizing different versions of the program under test. This is called the fault tolerance technique. These different versions then take the role of the test oracle. This approach is also not always reliable. Sometimes different versions of the program are not always developed independently. There is also a risk of faults which are common to all versions of a program appearing. Some have proposed that this method should be used in situations where one and only one version of a program is in existence. The problem with this technique is that it assumes the existence of an oracle and this assumption is not always correct. Also, given the nature of fault tolerance, this technique has the constraint that only the equality relationship is allowed.

Figure 1: Different Applications Graph
4. STEPS OF METAMORPHIC TESTING

To put it simply, metamorphic testing can be described in four main steps [3].

- The first step is to look for the metamorphic relations. These can be found from the specification of the software which is under test [4].
- The second step is to use some traditional test case selection methods and to generate the source test case and execute them.
- The third step is to construct and to execute the follow-up test case. This can be constructed from the source test cases based on metamorphic relations.
- The final step is to compare the results of both the source and the follow-up test cases. These should be compared against metamorphic relations.

In step one, it is highly recommended that testers concur with software users or developers [5]. This is to make sure that the identified metamorphic relationships are correct and necessary properties for the software under test. It is for this reason that domain-specific knowledge is required to fully understand the specification. Usually a metamorphic relation consists of two major parts. The first part is known as input relation. Input relation refers to the relationship between the inputs of source and follow up test cases. The second part is known as output relation. Output relation refers to the relationship which the outputs of source are supposed to hold with the follow up test cases [6].

4.1 Properties of good metamorphic relations: In order for metamorphic testing to be as effective as possible, it is crucial that the right relations be used. It is important that the relationships be designed with this in consideration [6]. In most case it is possible to identify a variety of metamorphic relations with different fault-detection capability. It is a good idea to use a variety of relations in order to test a program. One should use as many as possible in order for the test’s efficiency to be at its maximum.

5. APPROACHES OF TESTING

One must also be able to select relationships which are the most effective. Chen et al. talk about two approaches in selecting relationships, namely, the white box approach and the black box approach [7].

- The black approach refers to selecting a relationship based on theoretical knowledge of the problem.
- The white box approach refers to selecting a relationship based on the program structure. They concluded that the black box approach alone is not adequate for distinguishing good metamorphic relations and that in order to select good metamorphic relations one should use a white box approach and look at the algorithm under test.

Others such as Mayer and Guderlei have disagreed with this [8]. In Contrast, they have concluded that certain metamorphic relations have limited effectiveness such as equalities or linear equations and those close to the implementation strategy. In addition, they found good metamorphic relations to be strongly inspired by the semantics of the program under test. Good metamorphic relations should also try to make execution of the follow-up test...
Simarjeet Kaur, Mandeep Kaur:: A Review On Metamorphic Testing

case as different as possible from the source test case. As a matter of fact, Asrafi et al. have concluded that the higher the combined code coverage of the source and follow-up test case is, and the more the diversity of executions there are, will lead to more effective metamorphic relations[9]. Their study on two subject programs supported a strong correlation between coverage and fault-detection effectiveness on one of the programs. Cao et al. also performed a similar study. They analyzed the relation between fault-detection effectiveness of metamorphic relations and test case dissimilarity. In order to do this, they performed an extensive experiment. They found that with 83 faulty programs and 7 distance metrics between the execution profiles of source and follow-up test cases there was a strong and statistically significant correlation between the fault-detection capability of metamorphic relations and the distance among test cases. This distance occurred particularly when using branch coverage Manhattan distance [10]. It has also been shown that those metamorphic relations which derived from the components of a system are usually better at detecting faults in comparison to metamorphic relations which come from the whole system. Metamorphic relations which tend to target specific parts of the program under test are easier in regards to construction. They are also more constrained. This makes them more effective in detecting faults in contrast metamorphic relations at the system level. It is also important that metamorphic relations are described in a formal manner.

6. CHALLENGES OF METAMORPHIC TESTING

I. One challenge to the use of metamorphic testing is that there is a lack of guidelines for the construction of good metamorphic relations. Even though many authors have attempted to lay down guidelines, these guidelines often compliment or contradict each other. There is strong need for clear step by step guidelines on constructing metamorphic relations [9].

II. The biggest challenge perhaps is the generation of likely metamorphic relations. Here too, there has been research but authors have yet to find a concrete solution. There have been some breakthroughs; however, these have been mostly been restricted in scope. This research has mostly dealt with numerical programs. Right now there is a need for research regarding the generation of metamorphic relations in other domains as well as the use of different techniques for rule inference. There is also a lack of material on the synergies between the problem of generating metamorphic relations and the detection of program invariants [11].

III. Another challenge is the existence of an open problem regarding the combining of relationships. Many authors have written about the various pros of combining metamorphic relations. These combinations follow two different strategies. The first strategy consists of applying metamorphic relations in a chain style also known as IMT. The second strategy consists of the composing of metamorphic relations in order to construct new relations, also known as (CMR). There is still not enough literature which compares both of these strategies. Literature which talks about the heuristics to decide
when to use one or the other is much needed. Guidelines are needed to determine whether a given set of metamorphic relations can be combined and in which order[12].

IV. Another challenge is the automated generation of source test cases. When applying metamorphic testing, a majority of papers use either randomly generated or existing test suites as source. On the down side, it is evident that the source test cases usually influence the effectiveness of metamorphic relations. Today, there are many pending questions in regards to which cases make up the best possible source test cases and how such cases can be generated.

V. Another challenge is the lack of publicly available and maintained tools. Very few papers have been able to mention a tool for implementing various techniques which the papers present. This creates a problem for practitioners. If a practitioner wishes to apply metamorphic testing, they must implement their own techniques. This creates obstacles for these practitioners.

7. CONCLUSION
Metamorphic testing is an approach to the oracle problem in software testing. Metamorphic testing makes use of metamorphic relations for verifying test results as well as generating test cases. There are many applications of metamorphic testing there are also some challenges of this technique we will improve these in our research work.

8. REFERENCES
