بِسۡمِ ٱللَّهِ ٱلرَّحۡمَٰنِ ٱلرَّحِيمِ

Sudan University of Sciences and Technology

College of Graduate studies

Enhancing the Quality of Software Testing using a Metamorphic Testing Technique

تحسين جودة اختبار البرمجيات باستخدام طريقة الاختبار التحولي

a Thesis Submitted in Partial Fulfillment of the Requirements of M.Sc. Degree in Computer Science

Prepared By:

Sulieman Ibrahim Sulieman Bahar

Supervisor:

Dr. Mohammed Abdalla Osman Mukhtar

April 2021

Acknowledgment

I would like to express my thanks to Dr. Mohammed Abdalla for his valuable contributions and great efforts to advise me to complete the thesis.

Thanks go to staff members of Sudan University of Science and Technology as general and College of Computer Science and Information Technology as particular who exerted their time and effort to teach me useful science.

Finally I would like to thank my family for their enthusiastic help and endless encouragement.

الآيـــــة

بِسْمِ اللَّهِ الرَّحْمَنِ الرَّحِيمِ

((يَا أَيُّهَا الَّذِينَ آمَنُوا إِذَا قِيلَ لَكُمْ تَفَسَّحُوا فِي الْمَجَالِسِ فَافْسَحُوا يَفْسَحِ اللَّهُ لَكُمْ وَإِذَا قِيلَ انْشُزُوا فَانْشُزُوا يَرْفَعِ اللَّهُ الَّذِينَ آمَنُوا مِنْكُمْ وَالَّذِينَ أُوتُوا الْعِلْمَ دَرَجَاتٍ وَاللَّهُ بِمَا تَعْمَلُونَ خَبِيرٌ)) سورة المجادلة الاية (11).

Abstract

The software testing process plays an important role to improve the quality of the software product. The product or program which is free from errors greatly contributes to assure the quality of the software. An oracle in software testing is a person (tester) who does the process of testing. Therefor oracle problem arises due to the difficulty of determining the expected outcomes of selected test cases. A tester (oracle) may not always be available, or may be available but the process is too expensive and difficult to apply.

This research tried to reduce the effect of oracle problem during testing software. To overcome this problem metamorphic testing (MT) has been used to generate follow-up test case for multiple execution of program under test and verifying the result automatically.

The objectives of the research include analyzing the existing testing methods to reveal the gap of unavailability of solution of oracle problem, proposing an approach (MT) to enhance the quality of testing to solve the oracle problem and verify the proposed approach by applying it in selected case studies.

The researcher tried to use experimental method which explains the mechanism of work for (MT). Therefore JUNIT tool which supports MT was used to apply selected case studies.

The proposed method has been implemented using 3 case studies (trigonometric function, geometric shapes classification, booking web service). The obtained results showed a good enhancement in the testing process.

The importance of this research lies in overcoming oracle problem or alleviates it and thus, the research contributes to knowledge the domain by guiding researchers to use the metamorphic method because of its great advantages, as well as evaluating the effect of metamorphic method through empirical studies.

المستخلص

تلعب عملية اختبار البرمجيات دورًا مهمًا في تحسين جودة المنتج البرمجي. يساهم المنتج أو البرنامج الخالي من الأخطاء بشكل كبير في ضمان جودة البرمجيات. ان أوراكل في اختبار البرمجيات هو شخص (مختبِر) يقوم بعملية الاختبار. لذلك فإن مشكلة أوراكل هي صعوبة تحديد النتائج المتوقعة لحالات الاختبار المختارة. قد لا يكون المختبر (اوراكل) متاحا دائمًا ، أو قد يكون متاحًا ولكن عملية الاختبار مكلف للغاية ويصعب تطبيقه.

حاول هذا البحث تقليل تأثير مشكلة أوراكل أثناء اختبار البرمجيات. للتغلب على هذه المشكلة ، تم استخدام طريقة الاختبار التحولي لإنشاء حالة اختبار متابعة للتنفيذ المتعدد للبرنامج قيد الاختبار والتحقق من النتيجة تلقائيًا.

تشمل أهداف البحث تحليل طرق اختبار الحالية للكشف عن فجوة عدم التوفر لحل مشكلة أوراكل ، واقتراح طريقة (الاختبار التحولي) لتحسين جودة الاختبار لحل مشكلة أوراكل والتحقق من المنهج المقترح من خلال تطبيقه على دراسات الحالة المختارة.

حاول الباحث استخدام المنهج التجريبي الذي يشرح آلية عمل (الاختبار التحولي) لذلك تم استخدام أداة JUNIT التي تدعم تلك المنهجية لتطبيقه على دراسات الحالة المختارة.

الطريقة المقترحة تم تنفيذها باستخدام ثلاث دراسات حالة (دالة مثلثية ، تصنيف الاشكال الهندسية، خدمة الويب). أظهرت النتائج التي تم الحصول عليها تحسنًا جيدًا في عملية الاختبار.

تكمن أهمية هذا البحث في التغلب على مشكلة أوراكل أو التخفيف من حدتها ، وبالتالي يساهم البحث في اثراء المعرفة في المجال من خلال إرشاد الباحثين لاستخدام الاختبار التحولي لما لها من مزايا كبيرة ، فضلاً عن تقييم تأثيرها من خلال الدراسات التطبيقية.

Table of Contents

List of Figures:

List of Tables:

Chapter One

Introduction

1.1 Background

To improve the quality of the software product as general the software developers conducting several testing process. The importance of testing lies in assuring quality of software for the purpose of stakeholder's satisfaction. According to some studies the cost of testing represents 40% of total cost of development software project (money-effort-time).

To ensure the quality of the software, modern methods have been introduced to enhance the efficiency of software testing, reducing errors, increasing efficiency and reliability.

Software testing is a proven mechanism to assure the quality of software by revealing several types of errors like design errors, specification errors, code errors and input errors etc. Various testing methodologies have been introduced in the area of testing to assure the software quality and to identify the different types of software errors.

Metamorphic testing technique used to enhancing the quality of software through alleviating oracle problem.

In order to determine the expected results from the selected test cases, it is necessary to use the proposed method to overcome the oracle problem. The oracle test is sometimes not available and sometimes it is available, but the process is very expensive and difficult to implement.

Metamorphic testing it has own function known as metamorphic functional relations which in its role generates follow-up test cases and verify the outputs automatically to address oracle problem. Finally, this study comes to clarify an important aspect in software testing by using modern methods instead of traditional methods in order to improve the quality of the software product by reducing errors and increasing reliability. This study will provide reasonable results as a solution to the problem under study.

1.2 Problem Statement

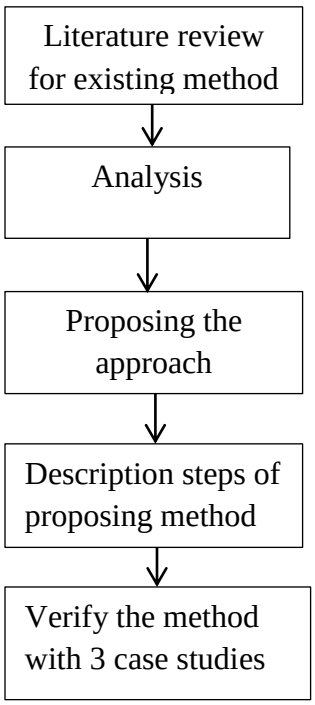
The oracle problem is the difficulty of determining or specifying the expected outcomes of selected test cases. A test oracle may not always be available, or it may be available but it is too expensive and difficult to apply. There is the need for proposing an approach to reduce the effect of oracle problem during testing process.

1.3Objectives

The objectives of the thesis are analyzing the existing methods to reveal the gap of unavailability for solution of oracle problem, proposing an approach to enhance the quality testing to solve the oracle problem and verify the proposed approach by applying it in selected case studies.

1.4 Research Methodology

The experimental method is used in the field of natural sciences as general, so it is used in the field of computer science to ensure the validity of the results of the processes under testing. In this thesis the researcher tried to use experimental method, and fitgure1.1 explains the steps of the research to achieve the objectives of the study.



Fitgure1.1: The Steps of the Research

1.5 Significance of the research

The importance of this thesis lies in overcoming the oracle problem or alleviates it, thus the research contributes in guiding researchers to use the metamorphic method because of its great advantages. As well as evaluate the effect of metamorphic method through empirical studies.

1.6 Thesis Organization

The thesis organized as follow: chapter one contains research background, research problem, motivation, research objectives, research methodology., significance of the research, contribution of research , and thesis organization.

Literature has been reviewed in chapter two, where many topics explored in details such as quality, testing, the Oracle problem and the metamorphic testing. As well as reviewing three case studies in some details. The chapter concluded with a review of fifteen studies that have a direct relationship to the topic of study.

Chapter three describes the methodology followed this research, which it has started by introduction, steps of metamorphic to solve the problem under study, the tool used to implement the selected case studies.

The result has been discussed in chapter four. It has started with introduction, then chart analysis for each results and result discussion for each implementation.

Introduction, conclusions and recommendations of the research have been provided in chapter five.

Chapter Tow

Literature review and related works

2.1 Introduction

The chapter explores two main topics; the first one is about literature review and second is related works, in term of first topic many topics have been explored such as software quality, quality factors, objectives of testing as general, oracle problem as well metamorphic testing technique.

In term of second topic; related works an exploration about fifteen studies which related to the oracle problem and test case generation and how to alleviate them by using proposed methodology.

2.2 Literature review

Software testing is a mainstream approach to software quality assurance and verification. Metamorphic testing is an approach to both test case generation and test result verification[1].

A metamorphic testing method has been proposed to test programs without the involvement of an oracle. It employs properties of the target function, known as metamorphic relations, to generate follow-up test cases and verify the outputs automatically. An “oracle” in software testing is a procedure by which testers can decide whether the output of the program under testing is correct. In some situations, however, the oracle is not available or too difficult to apply. This is known as the “oracle problem”. In other situations, the oracle is often the human tester who checks the testing result manually. The manual prediction and verification of program output greatly decreases the efficiency and increases the cost of testing[2].

Let **p** be a program implementing a specification **f**. Let **D** represent the input domain. Usually, it is impossible to do exhaustive testing to check whether **p**(t) = **f**(t) $\forall t \in \mathbf{D}$. As a result, a great amount of research in the literature of software testing has been devoted to the development of test case selection strategies, aiming at selecting those test cases that have a higher chance of detecting a failure. Let $\mathbf{T} = \{t1, t2, \ldots, tn\} \subset \mathbf{D}$ be the set of test cases generated according to some test case selection strategy, where $n \geq 1$. Running the program on these test cases, the tester

will check the outputs **p**(t1), p(t2), . . . , **p**(tn) against the expected results **f**(t1), **f**(t2), . . . , **f**(tn), respectively. If it is found that **p**(ti) 6= **f**(ti) for some i, where $1 \leq i \leq n$, then we say a "failure" is revealed and ti is a failure-causing input. Otherwise ti is a successful test case. The procedure through which the tester can decide whether **p**(ti) = **f**(ti) is called an oracle . For instance, let **f**(x, y) = x×y, the test case ti be {x = 3.2, y = 4.5}, and **p**(ti) = 14.4. The tester can verify this output either by manually calculating the product of 3.2×4.5 or using the inverse function to check whether 14.4/4.5 = 3.2, where the inverse can be done either manually or using a correct division program if available[3].

2.2.1 Software quality

Quality is a complex concept—it means different things to different people, and it is highly context dependent[4].

2.2.2 Five Views of Software Quality

There are five views of software quality, the first is transcendental View, and it is imagines quality as something that can be recognized but is difficult to define. The transcendental view is not specific to software quality alone but has been applied in other complex areas of everyday life. The second is user View; it perceives quality as fitness for purpose. According to this view, while evaluating the quality of a product, one must ask the key question: "Does the product satisfy user needs and expectations. The third is Manufacturing View; quality is understood as conformance to the specification. The quality level of a product is determined by the extent to which the product meets its specifications. The fourth is Product View; In this case, quality is viewed as tied to the inherent characteristics of the product. A product's inherent characteristics, that is, internal qualities, determine its external qualities. The fifth is Value-Based View; Quality, in this perspective, depends on the amount a customer is willing to pay for it[4].

2.2.3 Quality factors

There are many quality factors in software engineering but in this thesis some factors explained in the bellow table 2.1 which explains factors such as correctness, reliability, efficiency and integrity etc…

Table 2.1: McCall's Quality Factors and Criteria

Quality Factors	Definitions
Correctness	The extent to which a program satisfies its specifications and fulfills the user's mission objectives.
Reliability	The extent to which a program can be expected to perform its intended function with required precision.
Efficiency	The amount of computing resources and code required by a program to perform a function.
Integrity	The extent to which access to software or data by unauthorized persons can be controlled.
Usability	The effort required to learn, operate, prepare input, and interpret output of a program.
Maintainability	The effort required to locate and fix a defect in an operational program.
Testability	The effort required to test a program to ensure that it performs its intended functions.
Flexibility	The effort required to modify an operational program.
Portability	The effort required to transfer a program from one hardware and/or software environment to another.
Reusability	The extent to which parts of a software system can be reused in other applications.
Interoperability	The effort required to couple one system with another

2.2.4 Role of Testing

Software quality assessments is divided into two categories, the first one is static analysis it examines the code and reasons over all behaviors that might arise during run time[4]. The second is dynamic analysis which means actual program execution to expose possible program failure. Bellow figure 2.1 explains static and dynamic test in V-Model.

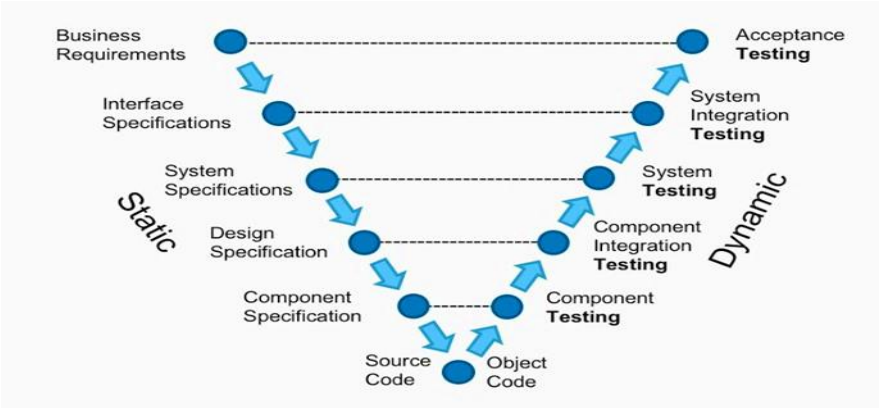


Figure 2.1 : static and dynamic test in V-Model[5]

2.2.5 Verification and Validation (V&V)

Verification means evaluation of software system that help in determining whether the product of a given development phases satisfy the requirements established before the start of that phase.

Validation means evaluation of software system that help in determining whether the product meets its intended use[4].

2.2.6 The Objectives of Testing

There are four objectives of testing the first is it does work; the programmer may want to test whether or not the unit works in normal circumstances. The programmer gets much confidence if the unit works to his or her satisfaction. The second objective is it does not work; here more tests are conducted with the objective of finding faults in the unit (or the system). Here, the idea is to try to make the unit (or the system) fail. The third objective is to reduce the risk of failures so a higher level objective of performing tests is to bring down the risk of failing to an acceptable level. The fourth objective is to reduce the cost of testing[4].

2.2.7 Test Case

Test case is a simple pair of <input, expected outcome> so an outcome of program execution may include value produced by the program, State Change and a sequence of values which must be interpreted together for the outcome to be valid[4].

According to IEEE standard test case is a set of inputs, execution conditions, and a pass/fail criterion.

2.2.8 Oracle concepts

An oracle is a procedure that determines what the correct behavior of a system should be for all input stimuli with which we wish to subject the system under test[6].

2.2.9 Basic concepts

Test case is a set of test inputs, execution conditions, and expected results developed for a particular objective. While test oracle is a principle or mechanism that helps you decide whether

the program passed the test. Verdict is a result (pass / fail / error / inconclusive…)[7]. Figure 2.2 explains basic concepts of testing.

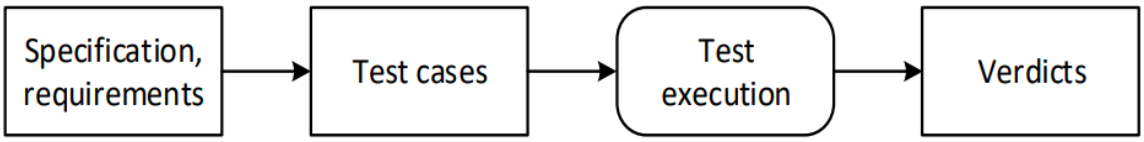


Figure 2.2: basic concepts of testing

2.2.10 Types of test oracles

There are four types of test oracle the first is Specified, this type of test oracle includes humans, textual specification and models (FSM, UML…). The second is implicit, this type of test oracle includes exceptions, crashes, security problems and robustness testing and fuzzing. The third is derived this type of test oracle includes previous program versions, different implementations (N-version programming) and assumptions, validity checks, invertible function[7].

The forth is handling the lack of oracles: in many cases no such artifact exists (i.e., there is no automated oracle D) and as such, a human tester is required to verify whether software behavior is correct given some stimuli[8]. Figure 2.3 explains the oracle types.

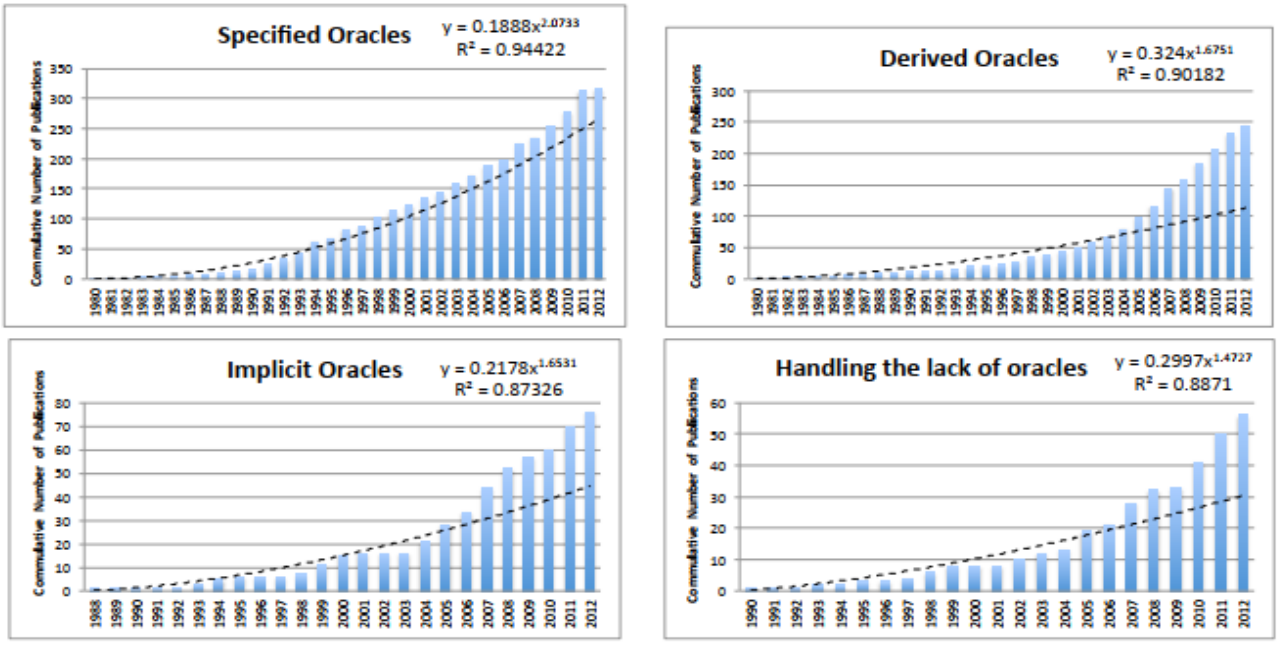


Figure 2.3: types of oracle

2.2.11Test Oracle

Is a mechanism, different from the program itself that can be used to test the accuracy of a program's output for test cases? Conceptually, we can consider testing a process in which test cases are given for testing and the program under test. The output of the two then compares to determine whether the program behaves correctly for test cases[9]. The figure 2.4 explains the basic idea of pseudo-oracles in (V&V).

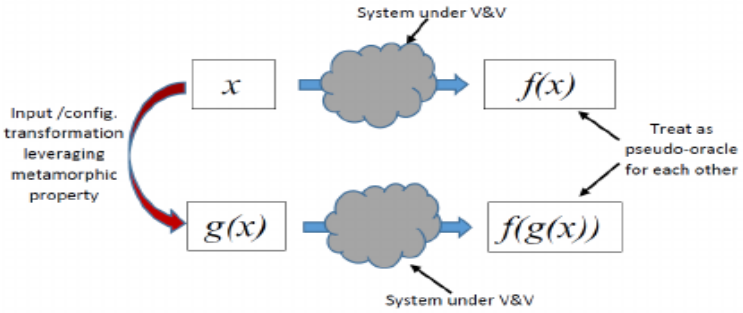


Figure 2.4 : the basic idea of pseudo-oracles in (V&V) [10]

2.2.12 Introduction to metamorphic testing technique

Metamorphic testing has been suggested by Chen et al. as a way of testing applications that do not have test oracles[11] ensuring that the software under test exhibits its expected "metamorphic properties.

Metamorphic testing (MT) has been developed to alleviate the oracle problem. Instead of focusing on the correctness of each individual output, MT checks the program against selected metamorphic relations (MRs). MR is a necessary property of the target function, and is a relation that involves multiple executions of the target function. MRs is identified based on knowledge of the problem domain, such as known properties of the target function/system or knowledge about the algorithms to be implemented. Each metamorphic test involves two or more executions of the program under test[12]. For ease of presentation, unless stated otherwise, we assume that each metamorphic test involves only two executions; the first execution, which is called the source execution, with its input called the source test case; and the next execution, called the follow-up execution with its input called the follow-up test case. If the outcomes of a source and its follow-up executions are found to violate an MR, the software under test must contain a fault.

Figure 2.5 explains relationships between several metamorphic testing conceptions

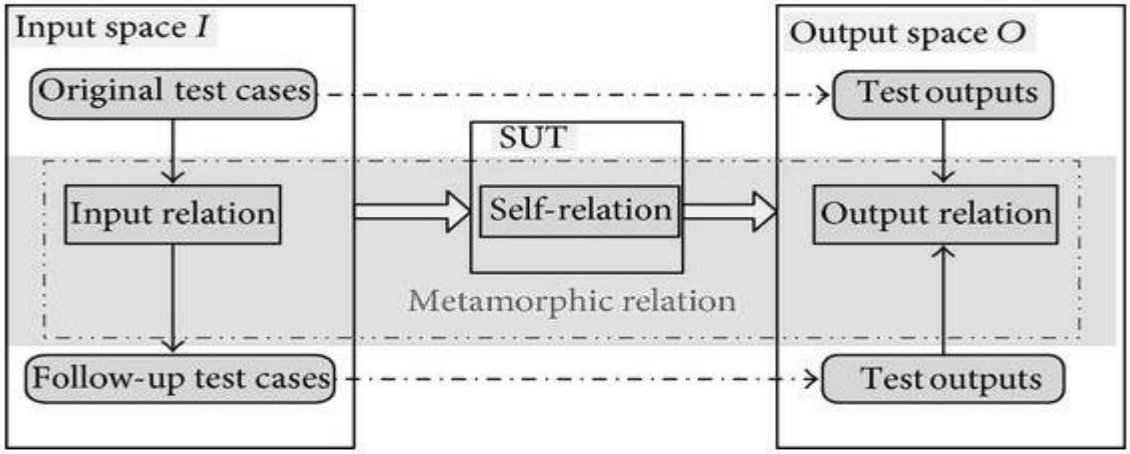


Figure 2.5: relationships between several metamorphic testing conceptions

2.2.12.1 Metamorphic steps

The basic steps for implementing metamorphic testing are as follows:

1- Some necessary properties of the software under test are identified (normally extracted from the specifications), and represented in the form of relations, referred to as metamorphic relations. Each metamorphic relation involves multiple test case inputs and their corresponding outputs.

2- Some test cases, referred to as the source test cases, are generated using traditional test case selection strategies.

3- New test cases, called the follow-up test cases, are constructed from the source test cases according to the metamorphic relations.

4- Both source and follow-up test cases are applied to the software under test.

5- The test case outputs are checked against the relevant metamorphic relations to confirm whether the relations are satisfied, or have been violated[13].

2.2.12 .2 Mathematical Properties

Many programs without test oracles rely on mathematical functions (i.e., those that take numerical input and/or produce numerical output), since the point of such programs is to implement an algorithm and perform calculations, the results of which cannot be known in advance; if they could, the program would not be necessary[11].

The classification of metamorphic properties as explained in the table 2.2.

Table 2.2: Classes of metamorphic properties

Additive	Increase (or decrease) numerical values by a constant
Multiplicative	Multiply numerical values by a constant
Permutative	Permute the order of elements in a set
Invertive	Take the inverse of each element in a set
Inclusive	Add a new element to a set
Exclusive	Remove an element from a set
Compositional	Create a set from some number of smaller sets

2.2.12 .3 Applications of metamorphic testing

Metamorphic testing is not limited to numerical programs only. In fact, metamorphic relations can be identified in almost every area like:

- Web services - Computer graphics - Embedded system simulations - Modeling
- Numerical analysis - Machine learning - Decision support - Bioinformatics
- Components - Compilers

The figure 2.6 bellow explains the percentage of MT using in different domains.

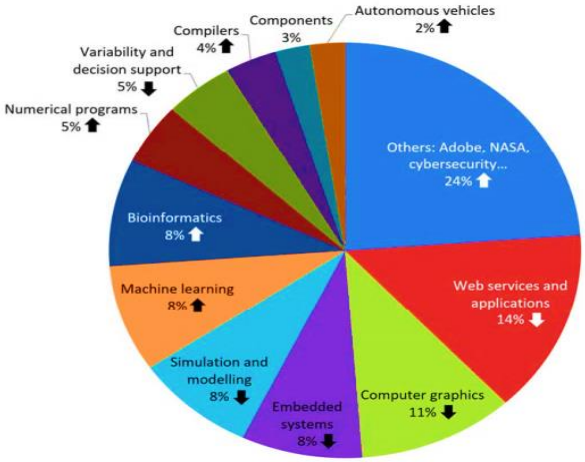


Figure 2.6: the percentage of using MT in different domains

2.2.13 other methods

Although metamorphic testing method has been adopted in different domains as a method of effective testing, there are other methods such model-based testing and Heuristics testing. Several researches confirmed the effectiveness of metamorphic testing than other testing methodologies.

2.2.13. 1 Model based-testing

Model based testing is a software testing technique where run time behavior of software under test is checked against predictions made by a model. A model is a description of a system's behavior. Behavior can be described in terms of input sequences, actions, conditions, output and flow of data from input to output.

2.2.13. 2 Heuristics and Oracles in Software Testing

When it comes to software testing, Oracles could also be the specifications and requirements of the software we test. So if there is a difference between oracles and the actual result, it is a bug because there is a deviation in the requirement or the expectations. Heuristics are the different approaches we apply to learn and test the software and its associated systems.

2.2.14 Experimental studies

To ensure the effectiveness of the metamorphic testing technique, 3 case studies have been applied to evaluate and verify the proposed method by compared to the traditional Oracle method.

Although there are a wide range of applications of the metamorphic testing method, the thesis limited to studying only three of the fields as a case study, the first is on trigonometric function sin(x), second case study is on the field of geometric shapes(triangle classification) and the third case study is on booking web services, which is specifically hotel reservation through the internet.

2.3 Related works

Chen, Y. Wang, Y. Guo, and M. Jiang in [14] proposed an MT approach for event sequences, which can be used to systematically test applications with rich business processes. They conducted three case studies in different domains to illustrate their approach. The experimental results demonstrate the feasibility and effectiveness of the approach. The results also confirm the previous findings that good MRs is those that make the executions as different as possible.

K. Yong in [15] aimed to address the cost problem in metamorphic testing by analyzing the various cost incurred in the metamorphic testing problems and their functional components.
Metamorphic testing technique has been proposed as an effective testing approach to detect failures in software with oracle problem.

W. K. Chan et al [16] argued that extends metamorphic testing into a user-oriented approach to software verification, validation and quality assessment, and conduct large scale empirical studies with a major web search engines: Google, Bings, and Chinese Bing, and Baidu. The objectives of the research were alleviating the oracle problem and challenges surrounding a lack of specifications when verifying, validating, and evaluating large and complex software system.
The empirical results showed that, firstly the approach can effectively detect various kinds of failures, secondly found that, the operational profiles have a significant impact on the quality of search.
W. K. Chan, T. Y. Chen, H. Lu, and S. S. Yau in[17] argued that During the testing of context-sensitive middleware-based software, the middleware identifies the current situation and invokes the appropriate functions of the application. Since the middleware remains active and the situation might continue to evolve, conclusion of some test cases may not be easily identified.
The research aimed to alleviate the above problems by making use a special kind of situation called checkpoints.
The research proposed to generate test cases that start at a checkpoint and end at another.
Based on a metamorphic approach they check the results of the test case to detect any contravention of such relations.

C. Aruna and R. S. R. Prasad in[18] introduced generic framework as metamorphic testing automation framework to address the selection of compatible base test suite, availability and

applicability analysis of metamorphic relation(MRs), automation metamorphic testing (MT) execution and follow up test cases generation problem.

This framework is an integrated environment with feasible metamorphic testing automation solutions to reveal the hidden bugs' information from software application.

Finally experimental results of metamorphic testing automation framework (MFAT) framework proven that, instead of using the individual testing methodologies separately, they can use them with MF as MFAT framework to identify the hidden bugs which may cause to big failure in future.

G. Dong, et al [19] argued that Two of the key challenges in software testing are the automated generation of test cases and the identification of failures by checking test output, these problem addressed by metamorphic testing technique.

The research aimed to present the technique and the results of a novel survey outlining its main trends and present some of successful applications to perform MT, they need to first identify some metamorphic relation (MRs) which are necessary properties among the inputs and outputs of multiple execution of the intended program functionality.

Finally there exist strong evidence of a rapidly growing interest in this topic from research community and industry.

C. Murphy, G. Kaiser, and L. Hu in [20] argued that Making machine learning applications dependable presents a particular challenge because conventional software testing process does not always apply. It's difficult to detect errors, faults, and defects.

The research used metamorphic approach to determine relationships that can be used to transform an input such that its new output will be predictable.

The research explored six metamorphic properties that may exist in many machine learning applications like additive, multiplicative, permutative, invertive, inclusive, and exclusive.

Finally the research found that metamorphic testing to be an efficient approach to test machine learning applications.

T. Y. Chen, F. Kuo, R. Merkel, and W. K. Tam in [21] argued that network modeling is a modeling technique for capacity planning studies of computer and communication system.

The main problem of the research was difficulty to know from computed outputs whether the computation of the modeling software is correct.

The research proposed metamorphic testing technique to testing queuing network modeling through a set of testing experiments in the java modeling tools.

The research results showed that different mutants were killed by different metamorphic relation, even though the source test cases were the same in each case.

J. Dunagan, et al [22] argued that Embedded system is the system where software is embedded inside hardware or electronic device to perform some function.

The problem of the research was to detect software failures of the wireless embedded system.

The research proposed metamorphic testing technique to address above problem.

The results showed that careful design of test environments and selection of system properties will enable us to trace back the cause of failures and help in fault diagnosis and debugging, moreover metamorphic testing technique enhancing quality of embedded software and quality of the wireless metering system.

W. K. Chan , S. C. Cheung§ and Karl R. P. H. Leung in [23] argued that Testing applications in service oriented architecture environments needs to deal with issues like the unknown communication partners until the service discovery, the imprecise black-box information of software components, and the potential existence of non-identical implementation of the same service.

The research aimed to exploit the benefits of the software oriented architecture (SOA) environments and metamorphic testing technique to alleviate issues.

The methodology of (MT) is support tester to apply the test cases for the unit test as the original test case for the integration test.

L. Xu, et al [24] argued that The automated extraction of information from feature models is a complex task that involves numerous analyses of operations, techniques and tools.

The research aimed to use metamorphic testing to automate the generation of test data for feature model analysis tools to overcoming the oracle problem.

An automated test data generator is presented and evaluated to show the feasibility of this approach.

The research presented a set of relations (so-called metamorphic relations) between feature models and the set of neighboring FMs together with their corresponding set of products are automatically generated and used for testing multiple analyses.

Complex FMs representing millions of products can be efficiently created by applying this process iteratively.

The evaluation results using mutation testing and real faults reveal that most faults can be automatically detected within a few second.

The results showed that the approach of metamorphic testing in the domain of automated analysis of feature models is efficient and effective in detecting most faults in a few seconds without the need for a human oracle.

A. Goffi in [25] argued that in software testing the role of test oracles is crucial; the quality of test oracles directly affects the effectiveness of the testing activity available techniques for generating test oracle are either effective but expensive or inexpensive but ineffective.

The research focused on the generation of cost-effective test oracles, multiple execution sequences perform the same, or almost the same action, this phenomenon is called intrinsic redundancy of software systems.

The research aimed to design and develop completely automated technique to generate test oracles by exploiting the intrinsic redundancy freely available on the software.

The recent survey classified test oracles in three categories: specified oracle, implicit oracle, and derived oracles.

Some challenges faced the research are: finding a suitable way to encode the equivalence sequences, executing the equivalence sequences, and equivalence check.

J. Ding, et al [26] reported on a novel use of metamorphic relations in machine learning, the author uses MR for the augmentation of the machine learning algorithms, and also reported on how MRs can enable enhancement to an image classification problem of image containing hidden visual markers (art codes).

Two different categories of images, and two MRs based on separation and conclusion was used to improve the performance of the classifier.

Experimental results showed that the MR-augmented classifier achieves better performance than the original classifier algorithms.

Finally it is very important to investigate this new approach direction including more case studies examining application of MRs to other well-known machine learning problems, such as face and object detection.

D. Peters and D. L. Parnas in [27], the researcher argued that an oracle is determined whether or not the results of a test execution are correct.

The research discussed ongoing work to produce a tool that will generate a test oracle from formal program documentation.

Tabular expressions are used to improve readability so that formal documentation can replace conventional documentations.

The research aimed at developing an automated test oracle generator tool that given a relational program specification.

The research concluded to that the generation of useful test oracle from relational program documentation id both feasible and practical.

J. Ding, T. Wu, J. Q. Lu, and X. Hu,in [28] argued that the metamorphic testing is an effective technique for system that do not have test oracle.

The research proposed self-checked metamorphic testing approach which integrates structural testing into metamorphic testing to detect subtle defects in a system implementation.

This approach, metamorphic testing results are further verified by test coverage information, which is automatically produced during the metamorphic testing, so the effectiveness of approach has been investigated through testing an image processing program.

In addition to checking the MRs the test coverage of function statement and definition use are automatically checked during the testing.

The results of the case study showed that checking the test coverage of metamorphic testing is an effective mechanism to insure the quality of MT. The flowing table 2.3 explains summary of related works. According to the summary 11 researches had have oracle problem solved by using metamorphic method while 3 studies had have test case generation problem and used (MT) to overcome this problem.

Table 2.3: Summary of related works

References	Oracle problem	Solution	MT	Year
J. Chen et al in [14]	Yes	Yes	Yes	2019
K. Yong in [15]	Yes	Yes	Yes	2015
W. K. Chan et al [16]	Yes	Alleviated	Yes	2015
W. K. Chan et al in[17]	Yes	Alleviated	Yes	2005
C. Aruna et al [18]	Yes	Yes	Yes	2017
G. Dong, et al [19]	Test case generation	Survey paper	Yes	2018
C. Murphy et al in [20]	Yes	Yes	Yes	-
J. Dunagan et al [22]	Yes	Yes	Yes	2009 - 2011
S. C. C et al In [23]	Yes	Yes	Yes	-
L. Xu et al [24]	Test case generation	Yes	Yes	-
A. Goffi et al in [25]	Yes	Yes	Yes	-
J. Ding et al [26]	Yes	Yes	Yes	2018
D. Peters et al [27]	Test case generation	Yes	Yes	-
J. Ding et al in [28]	Yes	Yes	Yes	2017

Chapter Three

Research Methodology

3.1 Introduction

In this chapter the researcher tried to explain the work of metamorphic testing technique, to get the intended results, and verify the performance of the method by explaining the steps of solution for selected case studies.

3.2 Case studies

For each case study metamorphic file has been created (java file) to pass it to JUNIT tool for testing either each test case pass or fail.

3.2.1 First case study (Trigonometric Function)

In order to find the value of the function sin (x), array created to store elements which generated randomly. These elements passed to the function as inputs to get the outputs each time and the validation of those outputs through the mechanism of the metamorphic testing method, and this method does not deal with the single execution but with the multiple execution of the program under test. These parameters which passed to the function are a set of test cases.

3.2.2 Second case study (Geometric Shapes Classification)

In the second case study, the idea of triangle classification program is it has consist of input as three natural numbers x, y, and z as the length of the side of a triangle. Its function is to classify triangle into equilateral (all sides the same length), or isosceles (two the same), or scalene (none the same), or to determine that the input does not represent an actual triangle when the summary of two parameters is not greater than the third. The following are the seed test cases.

•Test case t1: Input: (x=4, y=4, z=4), Expected output: Equilateral.

• Test case t2: Input: (x=4, y=4, z=6), Expected output: Isosceles.

• Test case t3: Input: (x=4, y=6, z=8), Expected output: Scalene.

• Test case t4: Input: (x=2, y=5, z=8), Expected output: Not a triangle

3.2.3 Third case study (Booking Web Service)

In this case study the idea of the program is enabling users to find potential lodgings according to their preferences. Firstly the program creates booking query object which includes destination room and setting beginning and ending of date then setting adult's number. Secondly the program creates follow up test case through setting budget and currency. Finally the program specifies metamorphic relations assertion. Metamorphic testing method in this case generate test case, execute them verifies the output automatically.

3.3 JUNIT tool

Unit testing is an important part in Test Driven Development (TDD) as it helps finding problems in the code as early as possible, especially when you make changes to the existing code you can run unit tests again to make sure that the changes do not break the application (regression). JUNIT is one of the most popular unit testing frameworks for Java development. JUNIT is supported by almost any Java IDEs and build tools, thus it is the default choice of programmers to test their code. Eclipse has very good support for JUNIT - the IDE is shipped with JUNIT as its default testing library. Thus writing and running unit tests with JUNIT in Eclipse is quick, easy and productive[29]. The figure 3.1 explains JUNIT 5 architecture.

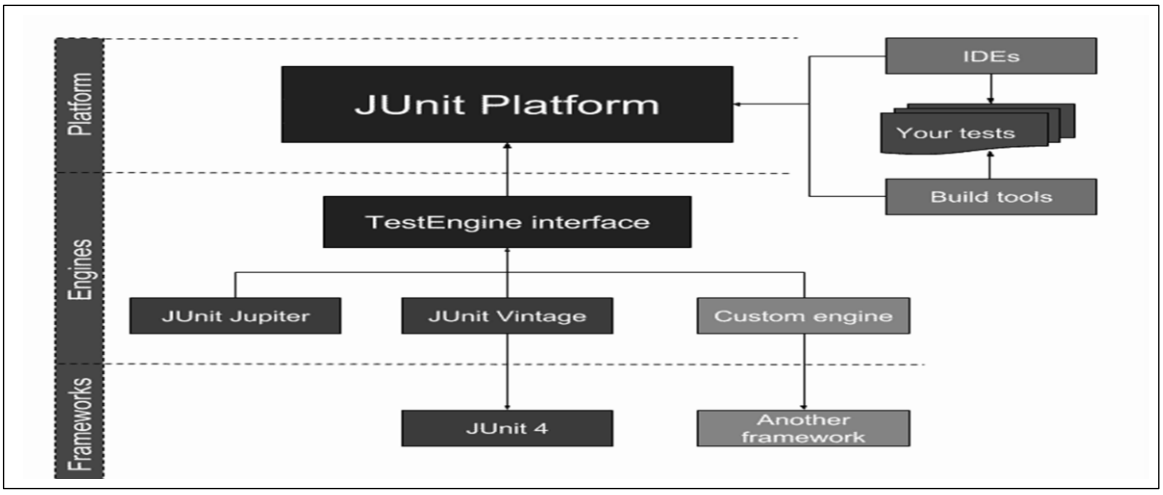


Figure 3.1: JUNIT 5 architecture

3.4 Description for the steps of proposed method

This section explored the steps of proposed method in details and how the proposed method enables the software developers (testers) to confirm the correctness of (MT).

3.4.1 Step one: identifying the metamorphic relation

In order to identifying metamorphic relation for software under test, there is the need for user specification. So metamorphic relation includes test suit inputs and their corresponding outputs.

3.4.2 Step two: test cases generating

The test case is divided into two types, source test cases which are generated using traditional test case selection strategies and follow-up test cases it is constructed from the source test cases according to the metamorphic relations.

3.4.3 Step three: test cases execution

Both source and follow-up test cases are applied to the software under test to know if test pass or fail.

3.4.4 Step four: outputs verifying

The test case outputs are checked against the relevant metamorphic relations to confirm whether the relations are satisfied, or have been violated. Therefor if expected outputs equal to actual value the test is pass otherwise the test fail.

3.5 Implementing the proposed method steps on 3 case studies

This section explored in details how to implement proposed method steps on 3 case studies to know the effectiveness of (MT).

3.5.1 Case study one (Trigonometric Function)

In this case study steps of proposed method have been implemented and the results showed.

3.5.1.1 Step1

When step1 has been implemented on case study1 the metamorphic relation identified according to problem domain and user specification.

3.5.1.2 Step2

When step2 has been implemented on case study1 the test cases generated according to test case selection strategies. The figure 3.2 bellow explains test case generation.

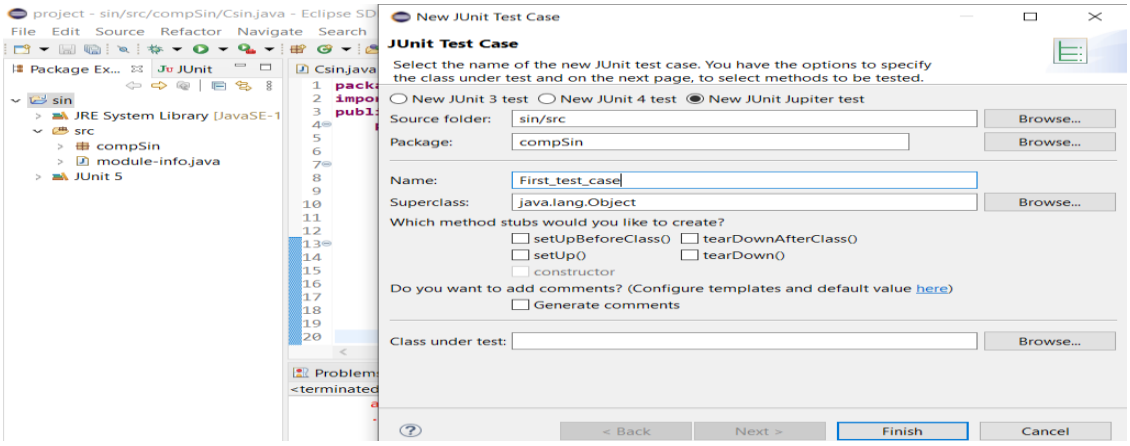


Figure 3.2: test case generation

3.5.1.3 Step3

When step3 of proposed method has been implemented on case study1 the results showed that no errors or failures as showed in figure 3.3: sin(x) test case implementation.

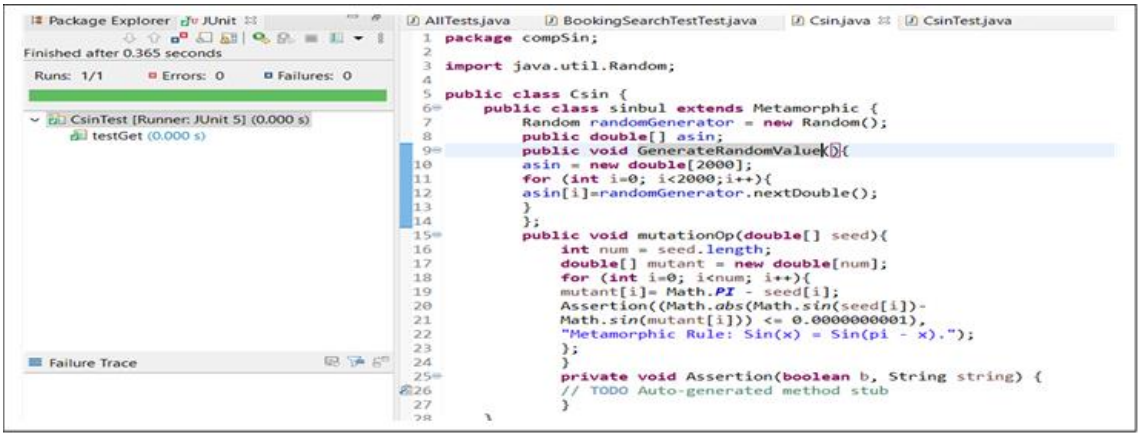


Figure 3.3: sin(x) test case implementation

The implementation of test suite in figure 4.3 sin(x)

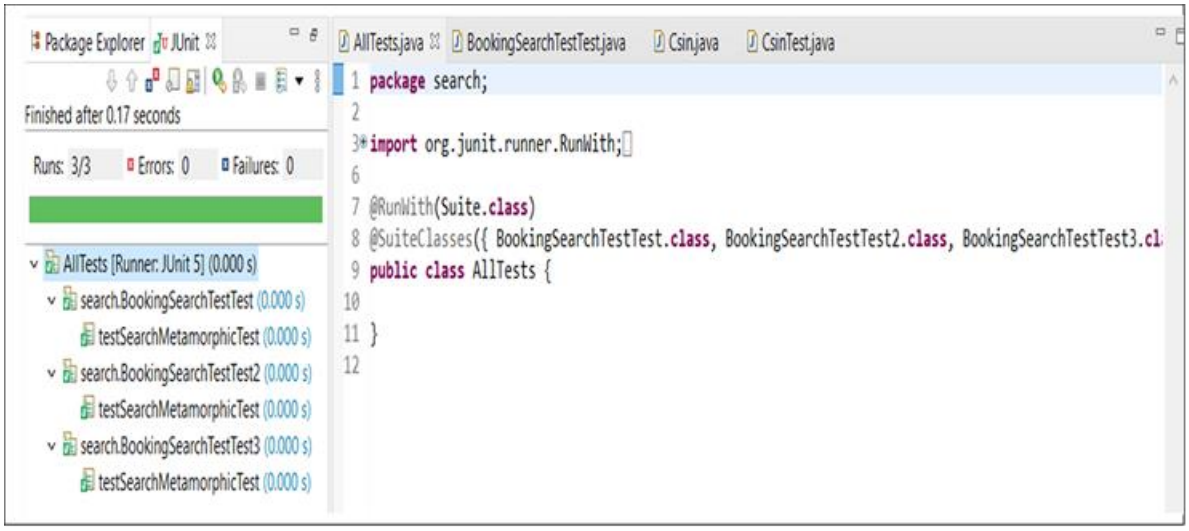


Figure 3.4: test suite sin(x) implementation

3.5.1.4 Step4

When step4 has been implemented on case study1 the results of testing checked according to metamorphic rules.

3.5.2 Case study2 (Geometric Shapes Classification)

In this case study steps of proposed method have been implemented and the results showed for each steps clarified.

3.5.2.1 Step1

When step1 of proposed method has been implemented on case study2 the metamorphic relation identified according to problem domain and user specification, and this steps extracted manually.

3.5.2.2 Step2

When step2 of proposed method has been implemented on case study2, test cases have been generated according to test case selection strategies, and there are two types of test cases the first one is source test case and the second is follow up test case. Figure 3.2 explained test case generation.

3.5.2.3 Step3

When step3 of proposed method has been implemented on case study2, the test case inputs implemented and showed results, no errors or failures occurred and this is strong evident that the proposed method more effective to detect or reveal different types of errors, therefor enhancing the quality of software under test. Figure 3.5 explains triangle test case implementation.

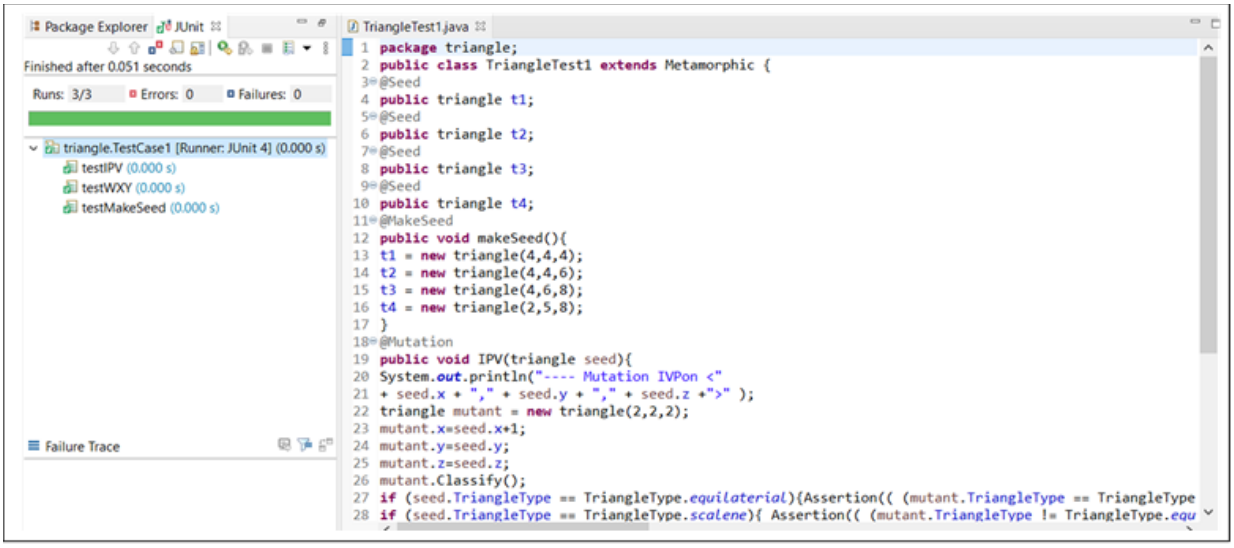


Figure 3.5: triangle test case implementation

The implementation of test suits the result of testing as follow in figure 3.6

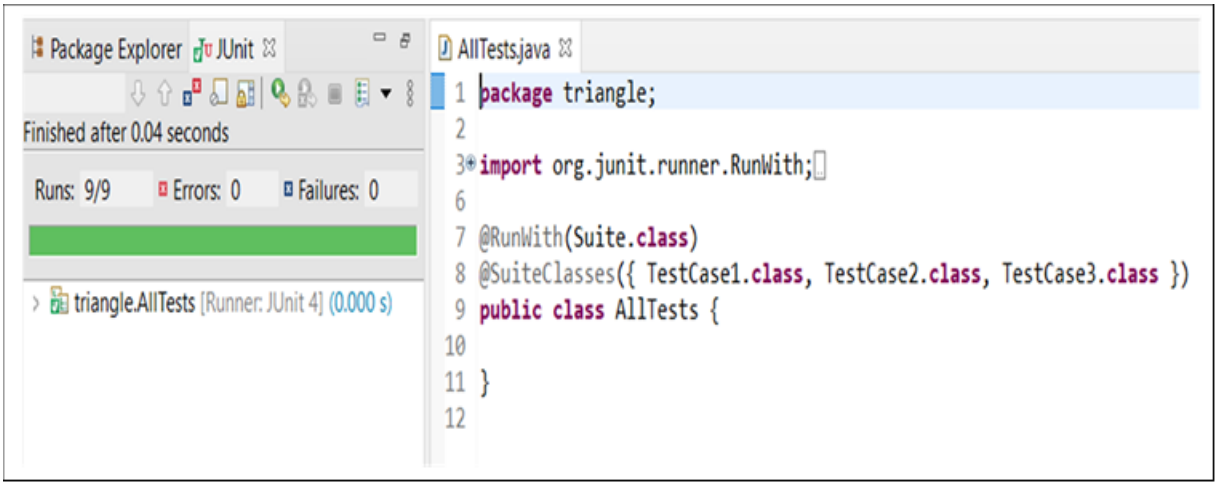


Figure 3.6: triangle test suite implementation

3.5.2.4 Step4

When step4 of proposed method has been implemented on case study2, the outputs of software under test verified against metamorphic relation, so the verification step checks if expected outputs equal to actual value. Then the test is pass otherwise the test is fail.

3.5.3. Case study3 (Booking Web Service)

In this case study steps of proposed method have been implemented and the results showed of each steps explained through practical application.

3.5.3.1 Step1

When step1 of proposed method has been implemented on case study3 the metamorphic relation identified according to problem domain or from user specification, and this steps extracted manually usually.

3.5.3.2 Step2

When step2 of proposed method has been implemented on case study3, test cases have been generated according to test case selection strategies, and there are two types of test cases the first one is source test case and the second is follow up test case which is extracted from source test case. Figure 3.2 explained test case generation.

3.5.3.3 Step3

When step3 of proposed method has been implemented on case study3, the test case inputs implemented and appeared results, that no errors or failures appeared and this is strong evident that the proposed method is more effective to reveal the different types of errors, therefor enhancing the quality of software under test. Figure 3.7 explains booking web service test case implementation.

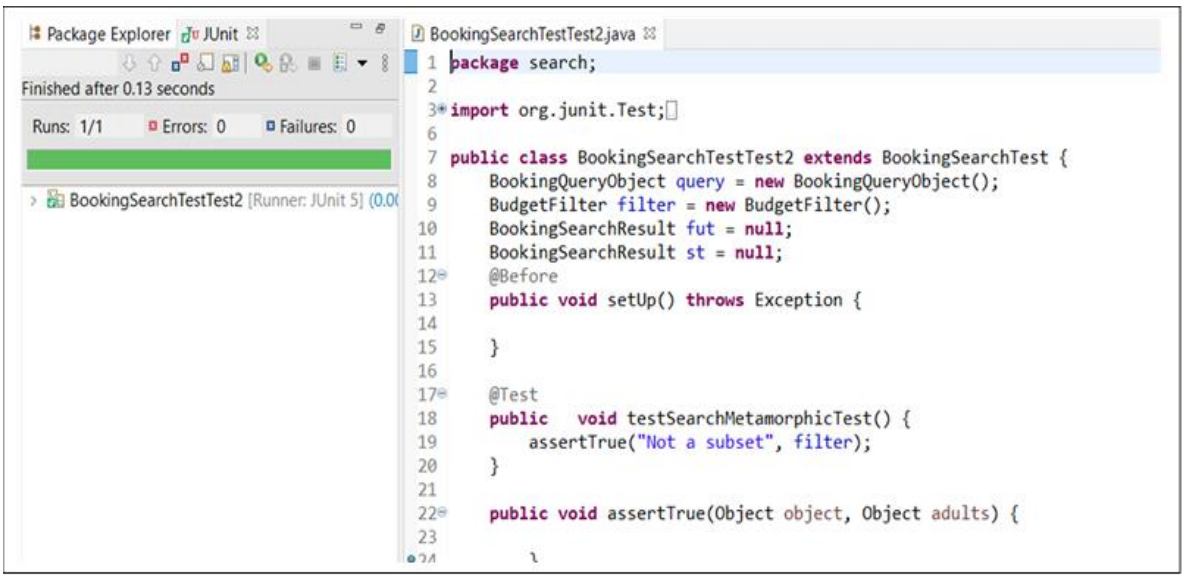


Figure 3.7: booking web service test case implementation

The implementation test suite on booking web service figure 3.8 explains it.

Figure 3.8: booking web service test suit implementation

3.5.3.4 Step4

When step 4 of proposed method has been implemented on case study3, the outputs of software under test verified and checked against metamorphic relation, so the verification step checks if expected outputs equal to actual value then the test is pass otherwise the test is fail.

Chapter Four

Results Discussion

4.1 Introduction

In this chapter, the researcher tried to discuss how to improve the quality of the software applications (programs under test), considering that one of the most important factors that helping to improve the quality is testing and verifying that they are free from functional and requirements errors.

Using the metamorphic test (MT) method to overcome the problem of test case oracle contributed greatly to improving the efficiency of the testing process and the accuracy of the results. Experimental results showed that the MT is very effective way to test program without involving an oracle.

4.2 First Case Study (trigonometric function)

Empirical execution of program in JUNIT tool showed that total runs (1), failures (0) and errors (0) for one test case while in test suits runs are(3) failures(0) and errors(0) Figures 1, 2, explain all aspects related to implementation of program by using JUNIT**.**

Figure 4.1 and figure 4.2 explain implementation of sin(x).

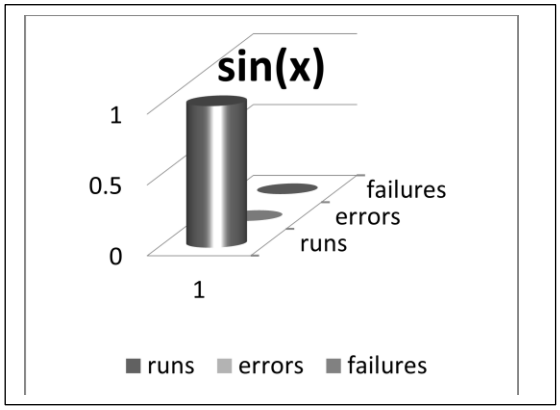


Figure 4.1: test case execution chart

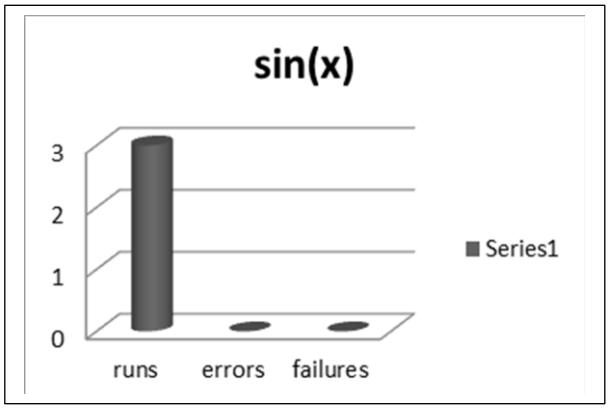


Figure 4.2: test suit execution chart

4.2.1 Discussions

The implementation process was seen free of errors, thus ensuring the smoothness and integrity of the code. On the other hand, the implementation result was devoid of failure, meaning there are no problems in the testing process, and this increases reliability, and thus it showed that the metamorphic method has a high efficiency to alleviate the problem of oracle.

The absence of errors and failures in the results of the testing process means the effectiveness of metamorphic method in ensuring the correctness of the outputs and thus contributes (MT) to improving the quality of the program.

4.3 Second Case Study (Geometric Shapes Classification)

Triangle has been taken as one of geometric shape classification. A set of test cases can be created and executed based on a set of pre-existing test cases, for example in the triangle classification program, there are three test cases: test case1, test case 2 and test case 3. When the test case has been started as shown in Figure (4.5), the number of parameters passed are (9), the errors are (0), and failure are (0) ,while only one test case executed in Figure (4.4). So that the total run becomes (3) with (0) errors and (0) failures. In Figure (4.4), the chart explains that total runs are (3), errors are (0) and failures are (3).

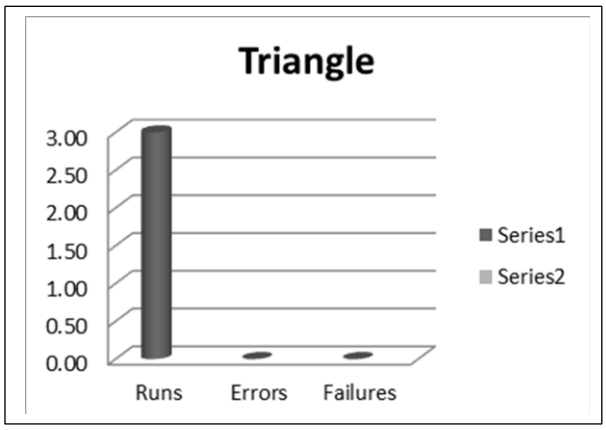


Figure 4.3: test cases implementation chart

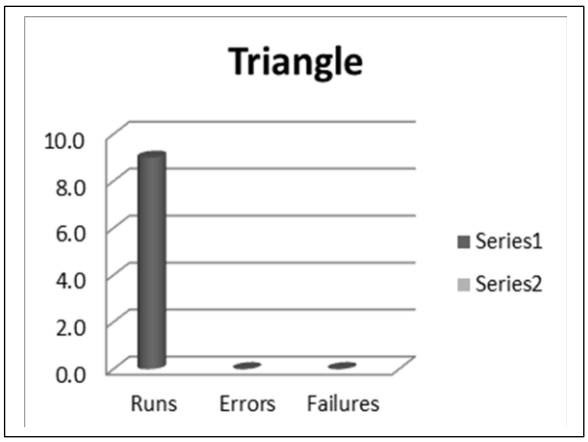


Figure 4.4: test suite implementation chart

The implementation of test case for the first time, it looks like the red pane which means there are some failures in test. Hence, it may need to rewrite them or fix the failures.

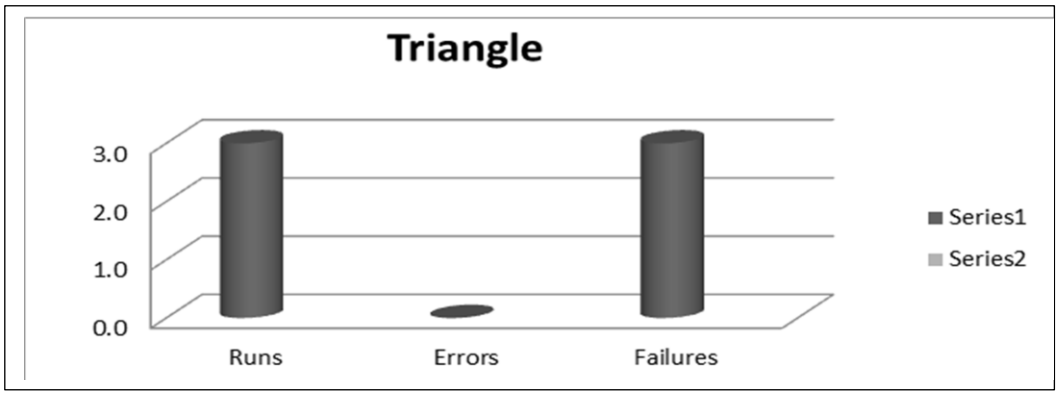


Figure 4.5: test case implementation chart

4.3.1 Discussion

From the charts, the fewer errors in the implementation, the more reliable and efficient the metamorphic method in software testing, and the success of all test cases means there is a strong logic code that can be relied upon.

The absence of errors and failures in the results of the testing process means the efficacy of the metamorphic method in ensuring the correctness of the outputs and thus (MT) contributes to improving the quality of the program.

Although there is a high implementation failure in Figure 4.9, it can be fixed by rewriting the test case.

4.4 Third Case study (Booking Web Service)

A test suit can be created and executed based on a set of pre-existing test cases. For example in the booking searching program, there are three test cases: test case1, test case 2 and test case 3.

When the test case has been started as shown in Figure (4.7), the number of parameters passed are (3), the errors are (0), and failure are (0), while only one test case executed in figure (4.6), error are(0) and failure are(0).

Figure 4.6 and figure 4.7 explain implementation of booking website.

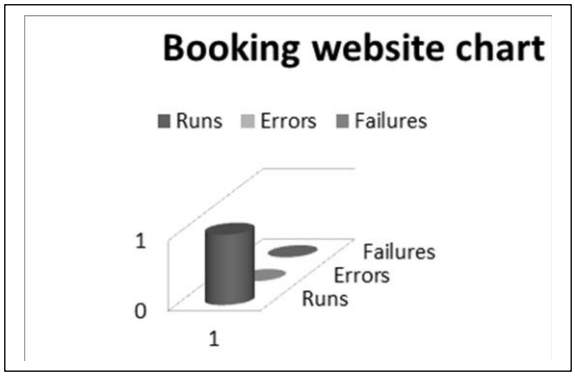


Figure 4.6: test case chart

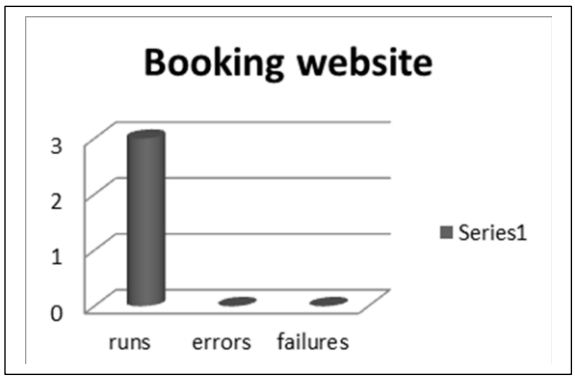


Figure 4.7: test suite chart

4.4.1 Discussion

In the charts, the fewer errors in the implementation, the more reliable and efficient the metamorphic method in software testing, and the success of all test cases means there is a strong logic code that can be relied upon. And then there are strong evident for using (MT) to solve the test case oracle problem.

The absence of errors and failures in the results of the testing process means the efficacy of the metamorphic method in ensuring the correctness of the outputs and thus (MT) contributes to improving the quality of the program.

4.5 General Discussion

Through the empirical study and the results it became clear that the testing process became smooth, as the successful implementation of each case study separately.

In the first case study, geometric shapes - one test case was executed and no error or failure was shown in the implementation. This is considered to be that the metamorphic method can be rely upon to overcome the oracle problem, as well as that the code is free of logical errors, but in return it can appear failures in execution, and this is normal due to the tool used, as some problems appear when creating a test case for the first time, and this requires rewriting the test case before implementation.

In the second case study - classification of geometric shapes, there is also a similarity in the results of the implementation, when executing one test case a very high failure occurred and it was fixed by rewriting the test case and this is normal as mentioned above, while a number of test cases were executed and no errors or failures appeared in execution.

In third case study- booking web service - one test case was executed, and the result of the implementation did not show any error or failure mentioned, which is evidence of the strength of the metamorphic testing in improving the quality of software testing. On the other hand when implementing a number of test cases also did not appear any error or failure in the result of the implementation.

Through the discussion and analysis above, we conclude that there is similarity in implementation between the three case studies, and appearance of failure in implementation is a natural thing due to the deficiency of the tool used in generating test cases, and despite some limitations of the metamorphic method, it is the most appropriate in software testing because of its characteristics which can link inputs and outputs to multiple relationships and validate results, thus contributing to improving the quality of the program under test.

Chapter Fife

Conclusion and Recommendations

5.1 Introduction

In this chapter; conclusion and recommendations explored, the final results explained and recommended some guides for postgraduate students to prepare their research, the researcher recommended some ideas that help to deal with this topic in the future time.

5.2 Conclusion

In conclusion, it concluded that the objectives of the study have been released and this is due to the successful implementation of the case studied and other reasons that mentioned in the previous. The researcher analyzed the defects of oracle and other traditional methods and their negative impact on the quality of the software and its inability to discover the hidden errors of the code, as well as the inability to test some systems due to unavailability of an oracle or the difficulty of implementing it.

As well as proposing the metamorphic method to improve the quality of the programs under test and to ensure the correctness of the outputs automatically as a solution to the oracle problem or mitigate it. In some cases where the metamorphic method was applied to the proposed case studies and the results proved a high efficiency of the proposed method in discovering errors and correcting them, which is considered a practical guide in achieving objectives of the study.

Finally, the effectiveness and efficiency of the method used was verified through the accuracy of implementation results for different test cases, and it can be relied upon in software testing for its significant contribution to improving software quality to the satisfaction of stakeholders.

5.3 Recommendations

In conclusion, it recommended developing the research to cover another side of the topic which includes using metamorphic technique in machine learning, computer graph, deep learning and modeling and simulation etc.

Addition to that developing this work and adding improvements up to be a large software project in the future times.

From the above, it recommended the following:

- Using another tool more effective than JUNIT because the limitation of the tool in test case generation and test case running
- Adding improvements as general
- Design interface for the project rather than dealing directly with code
- Searching for the best ways to generate test cases more efficiently
- Using another tool with JUNIT to integrate them in order to generate test case effectively (mutation testing)

References:

[1] Z. H. I. Q. Zhou, “Metamorphic Testing : A Review of Challenges and Opportunities,” vol. 000, no. 000, 2017, doi: 10.114.

[2] Z. Q. Zhou, D. Huang, T. Tse, Z. Yang, H. Huang, and T. Chen, “Metamorphic testing and its applications,” *Proc. 8th Int. Symp. Futur. Softw. Technol. (ISFST 2004)*, 2004.

[3] “Case Studies on the Selection of Useful Relations in Metamorphic Testing ⋆.pdf.” .

[4] E. Fuchs, *Quality: Theory and Practice*, vol. 65, no. 2. 1986.

[5] “V-model – Software Testing Watch This Video,” [Online]. Available: https://www.rogeriodasilva.com/v-model-software-testing/.

[6] M. Harman, P. Mcminn, M. Shahbaz, and S. Yoo, “A Comprehensive Survey of Trends in Oracles for Software Testing,” pp. 1–32.

[7] Z. Micskei and I. Majzik, “Model-based test generation Main topics of the course,” pp. 1–12.

[8] S. Segura, G. Fraser, A. B. Sanchez, and A. Ruiz-Cortes, “A Survey on Metamorphic Testing,” *IEEE Trans. Softw. Eng.*, vol. 42, no. 9, pp. 805–824, 2016, doi: 10.1109/TSE.2016.2532875.

[9] “geeks for gegeeks website.” https://www.geeksforgeeks.org/test-oracles/.

[10] “concepts of metamorphic,” [Online]. Available: https://www.researchgate.net/figure/The-concept-of-metamorphic-testing_fig1_280567781/download.

[11] C. Murphy, “Metamorphic Testing Techniques to Detect Defects in Applications without Test Oracles,” 2010.

[12] T. Y. Chen, F. C. Kuo, D. Towey, and Z. Q. Zhou, “Metamorphic testing: Applications and integration with other methods: Tutorial synopsis,” in *Proceedings - International Conference on Quality Software*, 2012, pp. 285–288, doi: 10.1109/QSIC.2012.21.

[13] H. Liu, F. Kuo, D. Towey, and T. Y. Chen, “How Effectively does Metamorphic Testing Alleviate the Oracle Problem ?,” pp. 1–21, 2013.

[14] J. Chen, Y. Wang, Y. Guo, and M. Jiang, *A metamorphic testing approach for event sequences*, vol. 14, no. 2. 2019.

[15] S. K. Yong, “Cost-effective Metamorphic Testing Techniques for Failure Detection in Software with Oracle Problem,” 2015.

[16] Z. Q. Zhou, S. Xiang, and T. Y. Chen, “Metamorphic Testing for Software Quality Assessment : A Study of Search Engines,” no. January, 2015, doi: 10.1109/TSE.2015.2478001.

[17] W. K. Chan, T. Y. Chen, H. Lu, and S. S. Yau, “A Metamorphic Approach to Integration Testing of Context-Sensitive Middleware-Based Applications ∗,” 2005.

[18] C. Aruna and R. S. R. Prasad, “MTAF : A Testing Framework for Metamorphic Testing Automation MTAF : A Testing Framework for Metamorphic Testing Automation,” no. September 2015, 2017.

[19] S. Segura, “Metamorphic Testing 20 Years Later : A Hands-on Introduction,” pp. 3–6, 2018.

[20] C. Murphy, G. Kaiser, and L. Hu, “Properties of Machine Learning Applications for Use in Metamorphic Testing.”

[21] T. Y. Chen, F. Kuo, R. Merkel, and W. K. Tam, “Testing an Open Source Suite for Open Queuing Network Modelling Using Metamorphic Testing Technique Testing an Open Source Suite for Open Queuing Network Modelling using Metamorphic Testing

Technique," no. April 2014, 2009, doi: 10.1109/ICECCS.2009.28.
[22] F. Kuo, T. Y. Chen, and W. K. Tam, "Testing Embedded Software by Metamorphic Testing : a Wireless Metering System Case Study," pp. 291–294, 2011.
[23] S. C. C. and K. R. P. H. L. W. K. Chan †‡, "Towards a Metamorphic Testing Methodology for Service-Oriented Software Applications ∗.pdf." .
[24] "Automated Metamorphic Testing on the Analyses of Feature Models ☆.pdf." .
[25] A. Goffi, "Automatic Generation of Cost-Effective Test Oracles Categories and Subject Descriptors."
[26] L. Xu, D. Towey, A. P. French, S. Benford, and T. Y. Chen, "Enhancing Supervised Classifications with Metamorphic Relations," 2018.
[27] D. Peters and D. L. Parnas, "Generating a Test Oracle from Program Documentation work in progress."
[28] J. Ding, T. Wu, J. Q. Lu, and X. Hu, "Self-Checked Metamorphic Testing of an Image Processing Program Self-Checked Metamorphic Testing of an Image Processing Program," no. August, 2017, doi: 10.1109/SSIRI.2010.25.
[29] "eclips," [Online]. Available: https://www.codejava.net/testing/junit-tutorial-for-beginner-with-eclipse.