Imperial Journal of Interdisciplinary Research (IJIR)
Vol-3, Issue-3, 2017
ISSN: 2454-1362, http://www.onlinejournal.in

Generation of Source Test Cases in Metamorphic Testing

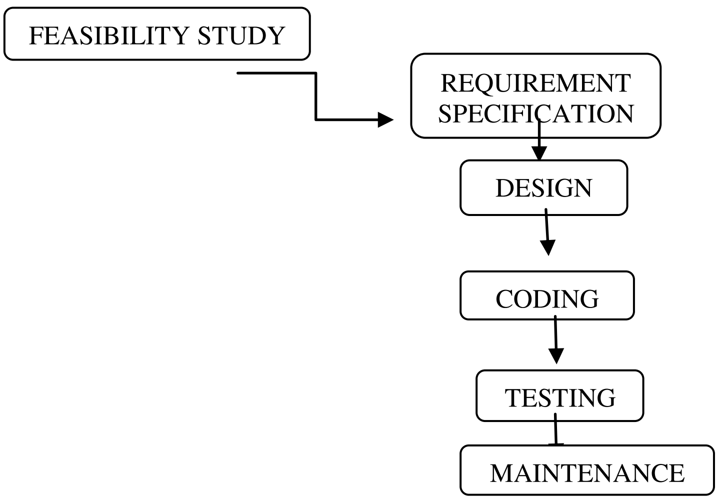
Dr. Amit Verma[1] & Rupinderjeet Kaur[2]
Dept. Computer Science and Engineering, CEC, Landran , Mohali, India

Abstract- A test oracle is a mechanism that figures out if a test execution uncovers a deficiency, regularly by contrasting the watched program yield with the expected yield. Metamorphic Testing is one of the technique of software testing, that gives an flipside to test the program without any involvement of test oracle. The Potency of Metamorphic Relations can play a vital role in the testing process. For Proper maintenance of software, testing is very necessary and testing can be carried out by applying some input in the form of source data and then we record the output. This recorded output will be compared with the expected output, but for performing this procedure there will be a requirement for an oracle, which is not easily available due to its expensiveness. For this purpose, to eliminate the need of an oracle, technique named metamorphic testing is introduced. In Metamorphic Testing, metamorphic relations are of great importance. Since the presentation of such metamorphic relations in 1998, numerous commitments on metamorphic testing have been made, and the method has seen effective applications in an assortment of spaces, going from web administrations to PC representation .The term Metamorphic Relations apparently alludes to the "transformation" of test inputs and yields. The source test cases i.e. input are generated from these metamorphic relations and then from this generated source test cases, follow-up test cases are formed. The Metamorphic testing requires the utilization of source test cases that serve as seed for the generation of follow-up test cases. Source test cases are the info esteem which we figure in initial step. Testing can be led to make the product bug free. Software testing is exceptionally fundamental part in the advancement of programming. For increasing the reliability of the software, testing is important. In this paper I intend to explain the main concept of software engineering as my thesis topic is in this domain i.e metamorphic testing which is one of the technique of software testing.Testing is necessay to make a software bug free and more reliable. This review paper spotlight the principle idea of metamorphic testing and concept of generating source test cases automatically by using soft computing technique.

Index Terms—*,Software Testing, Metamorphic Testing, Metamorphic relations, Test Oracle, Test cases.*

I. INTRODUCTION

Few centuries ago, when people uses computers at that time Software's were expensive, mainly led to bugs and errors in the system so to overcome this problem, Software Engineering is introduced in 1950s. Software engineering is a study of engineering to the design, development ,implementation and maintenance of a software in a systematic way. In other words, Software engineering is to study the existing information and knowledge from an experienced person in the further development of software so that it should be reliable and efficient. In software .engineering there is Software Development Life Cycle (SDLC).In SDLC ,we have 6 phases:-
1.Feasibility Study
2.Requirement gathering and analysis.
3.Design.
4. Coding.
5.Testing.
6.Maintenance.



Feasibility study:-
It is a study to plan for the things that are required for the development of the product in terms of cost and profit.

Requirement Gathering and Analysis:-
In this phase information is collected that is required for the product and analyzed properly according to the requirement of the customer. For example:-
Q1.What is the problem domain ?
Q2.How it can be solved ?

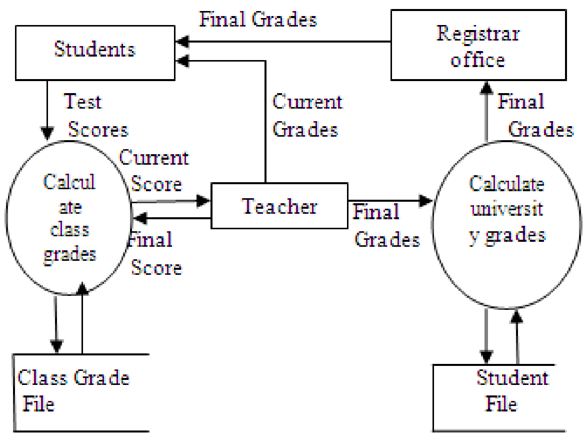
Design:-
In this phase the system and software design is prepared from the requirement specifications. System Design helps in specifying hardware and system requirements and also helps in defining overall system architecture. The system design specifications serve as input for the next phase of the model.

Coding :-
From design documents, work is divided into modules/units, actual coding is started. This one is the longest phase of SDLC.

Testing:-
Testing plays a vital role in the development process. Testing is done to ensure that the system developed is free of bugs and errors.

Maintenance:-
The Proper Maintenance should be required for the system so that it should be more reliable. It includes:-
1.Corrective:-The System should be free of any bugs.
2.Perfective:-
3.Adaptive:-System should be portable i.e can run on any platform.

Now we will consider one example by making DFD of SDLC on Teaching :-



SYMBOLS USED IN DFD ARE AS FOLLOW:-

▭ Entity that supplies or receives data.

○ Process

→ Data Flows

⊏ Data stores

# Factors Affecting the Software engineering:-
1.Risk Analysis
2.Cost effectiveness
3.Product Quality

SOFTWARE TESTING
Software Testing is an extremely urgent step towards guaranteeing nature of the product. It is a kind of examination which is directed on the product created with the goal of finding software bugs. Testing begins at the part level and afterward go towards integration of the entire framework.
A few software testing methodologies:-
1.To do Testing adequately, programming group ought to sort out authority gatherings and examinations. By this, a ton of mistakes will be distinguished before real testing starts.
2.Testing begins at the part level and afterward go towards integration of the entire framework.
3.Various testing techniques are accessible which can be applied in various situations.

SOFTWARE TESTING TECHNIQUES	
White-box testing	Based on external specification.
Black-box testing	Based on internal specification.
Unit testing	Based on module generation. It includes stub and driver.
Integration testing	Based on module interference.
System testing	Based on reliability of system.
Metamorphic testing	Based on potency to eliminate the oracle problem.

TYPES OF SOFTWARE TESTING:-

1. Black-box Testing:-
Test cases are designed using only the Functional specification of the software. It does not require any knowledge of design and code.
2. White-box Testing:-
It requires the knowledge of design and code. It tests the internal structure and is opposite to Black-box Testing.
3. Unit Testing:-
Unit testing also called component testing is meant for testing the individual functionality of the smallest module generated.
4. Integration Testing:-
This testing is done to check the interfaces between the components.

5. System Testing:-
It is needed so that the system should be error free and should be according to the requirements of the customer.
6. Metamorphic Testing:-
It is One of the Technique of Software testing to test the program without the need of an oracle.

Software testing assumes an essential part in the improvement of programming for distinguishing the issues in the system. When we apply typical testing to the framework in some cases it neglects to identify a few bugs in the framework and by this frameworks gets to be untrustworthy, so to cover these revealed deficiencies we present a technique known as "Metamorphic Testing". At the point when a project is executed we give an input to get the yield. To check project's consistency we have to check whether the testing connected to the framework "passed" or "fizzled", however for this checking reason we require one component which is called as "oracle", which is not clearly accessible. To beat this "oracle issue" we have to apply Metamorphic testing. Metamorphic Testing is a creative way to deal with oracle issue, it is a procedure connected for identifying the revealed flaws in a project.

Identity relations are a surely understood idea in testing, and have been utilized even before the presentation of metamorphic relations. The term metamorphic relation apparently alludes to the "transformation" of test inputs and yields: The general idea of metamorphic testing is that a current source experiment (x1) is changed ("transformed") into a subsequent experiment (x2), such that a conceptual relation (R) over source and catch up test inputs and yields can be checked. On the off chance that the relation does not hold, a flaw has been recognized.

In this paper, we use the term metamorphic test case to refer to a pair of source test case and its follow–up test case.

The primary procedure for the use of metamorphic testing can be outlined as follows:-
1) Construction of metamorphic relations:-
Pinpoint the essential properties of the project under test and that properties will be spoken to as metamorphic relations among numerous experiment inputs and their normal yields, alongside some strategy to create a follow–up test case that depends on a source test case.

2) Generation of source test cases:-
Generate or select a set of source test cases for the program under test using any traditional testing technique (e.g., random testing).
3) Execution of metamorphic test cases:-
By utilizing the metamorphic relations follow–up test cases are generated, then execution of source and follow–up test cases held, and afterward we check the relations. On the off chance that the yields of a source test case and its follow–up test case damage the metamorphic relation, the transformative experiment is said to have fizzled, demonstrating that the system under test contains a bug.

$$\begin{tikzcd}\boxed{\text{MR1}} \arrow[r, Rightarrow] & \boxed{\text{MR2}}\end{tikzcd}$$

(a) source test case (b)follow-up test case

Metamorphic testing is normally performed according to the following steps :-
STEP 1: Pinpoint the imperative relations among the output of the program under execution.
STEP 2: Generate the source test case using some traditional test case selection methods and execute them.
STEP 3: Construct the follow-up test case from the source test cases rely on metamorphic relations, and execute them.
STEP 4: Collate the outcomes of source and follow-up test cases against metamorphic relations.

By using these four step, we can evaluate the metamorphic testing, by generating the source test cases and constructing the follow-up test cases.
In the event that source test cases are created naturally, metamorphic testing can be viewed as a automated testing strategy empowering full test mechanization, i.e., input generation and yield checking.

CONSTRUCTION OF METAMORPHIC RELATIONS:-

Constructing metamorphic relations is typically a manual task that demands careful learning of the system under test. We represent some proposed alternative ways to create metamorphic relations, either by combining existing relations or generating them automatically.

APPLYING METAMORPHIC RELATIONS IN A :-

CHAIN STYLE (IMT) COMPOSING METAMORPHIC RELATIONS (CMR)
CHAIN STYLE:-
If there are two relations and we have one source test case and one follow-up test case. This follow-up test case is generated from source test case then that follow-up test case is used as source test case of the other relation.
If we have MR(a) and MR(b)
STEP1 Source test case is used to generate follow up test case of MR(a)
STEP2 Follow-up test case of MR(a) is used as source test case of MR(b)

COMPOSING METAMORPHIC RELATIONS:-

In this we construct new metamorphic relations by combining several existing relations. The new relation should impose all the properties of the existing one.

STEP1.Let MR(x)and MR(y) are two metamorphic relations, MR(y) is compositable to MR(x) if for any source test case T for MR(x) its corresponding follow-up test case is

Fx(T). Fx(T)is source test case for MR(y).

STEP2.Let MR(x)and MR(y) be two relations.MR(y)is compositable to MR(x).

MR(xy) is composition of MR(x) and MR(y).
Source test case for MR(xy) is T
Follow-up test case is F(xy)(T)=F(y(Fx(T))

By composing the metamorphic relations we have one source test case and one final follow-up test case for MR(xy).

Suppose we have two relations i.e.

MR(x)
MR(y)

The composition of MR(x) and MR(y) is

$\boxed{\textbf{MR(xy)}}$

In IMT(Iterative Metamorphic Testing)Metamorphic relations are executed in a chain style. In this execution takes place in two steps. Firstly, transformation is applied to the inputs of source test case to generate follow-up test case. Secondly, source test cases and follow-up test cases are executed. In this paper, author execute the metamorphic relations in a chain by reusing the follow-up test case of each metamorphic relations as a source test case of the next metamorphic relation.

STEP1:-

SOURCE TEST CASE

$\boxed{\text{MR1}} \Longrightarrow \boxed{\text{MR2}}$

FOLLOW-UP TEST CASE IS GENERATED FOR MR1

STEP2:-

SOURCE TEST CASE(follow-up test
Of MR1 is reused)

$\Longrightarrow$

FOLLOW-UP TEST CASE IS GENERATED FOR MR2

STEP3:-

SOURCE TEST CASE(follow-up test
Case of MR2 is reused)

$\boxed{\text{MR3}} \Longrightarrow \boxed{\text{MR4}}$

Fig1.Metamorphic testing in a chain style

We can execute the metamorphic relations in a chain style .

For example :

MR1 →MR2→MR3→MR4

Follow-up test case of MR1 is used as source test case of MR2

And follow-up test case of MR2 is used as source test case of

MR2 and so on.

Iterative Metamorphic testing is more efficient than any other testing and also generate test cases with a great capability for detecting faults.

TECHNIQUES	DESCRIPTION
CMR (composition of metamorphic relations)	Composing Metamorphic Relations, in this two metamorphic relations are combined to construct Metamorphic relation. MR(A) and MR(B) Composition of MR(A) and MR(B) is MR(AB).
IMT (iterative metamorphic testing)	In this source test case of one metamorphic relation is used as follow-up test case of other metamorphic relation. MR(A) → MR(B) → MR(C)

It describes techniques to illustrate metamorphic relations

GENERATION OF SOURCE TEST CASES:-

In previous work test cases are generated randomly or existing test suits are used as source tests when applying metamorphic testing. Metamorphic testing requires the use of source test cases that serve as a seed for the generation of follow-up test cases. In thesis work main topic i.e. Generation of Source test cases. we generate the source test cases by using one algorithm named “FLOWER POLLINATION”. The flower pollination algorithm (FPA) was developed by Xin-She Yang in 2012, inspired by the pollination process of flowering plants. FPA has been extended to multiobjective optimization with promising results.

EXECUTION OF METAMORPHIC TEST CASES:-

The execution of a metamorphic test case is typically performed in two steps.

First, a follow–up test case is generated by applying a transformation to the inputs of a source test case. Second, source and follow–up test cases are executed, checking whether their outputs violate the relation. Our future work is to execute the test cases by using the algorithm i.e Flower Pollination Algorithm.

II.REVIEW

Till now, many authors have explored their research on the topic ”metamorphic testing”. Metamorphic Testing has been applied to many domains, But in this section we will spotlight
the papers proposing elective techniques for the generation of source test cases.

Gotlieb and Botella [1] displayed a structure named Automated Metamorphic Testing (AMT) to naturally produce test data for metamorphic relations. Given the source code of a project written in a subset of C and a metamorphic relation, AMT tries to discover test cases that damage the relation.

Chen et al. [2] looked at the viability of "special values" and random testing as source test cases for metamorphic testing. Special values are test inputs for which the expected yield is understood (e.g., sin(360) = 0). Since test cases with unique qualities must be manually constructed ,we consider them as manual testing. The authors found that manual and metamorphic testing are integral strategies, however they likewise take note of that random testing has the point of interest of having the capacity to give much bigger test information sets.

Segura et al. [3] analyzed the adequacy of random testing and a manually planned test suite as the source test cases for metamorphic testing, and their outcomes additionally demonstrated that random source test cases are more successful at identifying flaws than manually planned source test cases in all the projects. Despite the fact that this recommends random testing is more compelling, there are additionally signs that consolidating random testing with manual tests might be far and away superior and segura et.al observed that consolidating manual tests with random tests prompts shortcomings being recognized speedier than when simply utilizing random tests.

Batra and Sengupta [4] displayed a Genetic algorithm for the determination of source test cases amplifying the ways crossed in the project under test. The objective is to produce a little however exceedingly viable arrangement of source test cases. Their algorithm was assessed by producing source test cases for a few metamorphic relations in a small C program for deciding the kind of a triangle, where 4 mutants were produced and executed.

In a related work, Chen et al. [5] tended to the same issue from a black–box point of view. They proposed parceling the input space of the program under test into comparability classes, in which the system is required to handle the inputs similarly. At that point, they proposed a algorithm to choose test cases that spread those comparability classes. Assessment on the triangle program recommends that their calculation can produce a small set of source test cases with high detect rate.

Deepika generated the source test case which is taken as an input by using soft computing technique i.e. Genetic Algorithm.

Dong and Zhang [6] exhibited a technique for the development of metamorphic relations and their comparing source test cases using symbolic execution. The technique first investigates the source code of the project to decide the symbolic inputs that bring about the execution of each path. At that point, the symbolic inputs are manually reviewed and used to guide the development of metamorphic relations that can practice every path of the project. At long last, source test cases are produced by supplanting the symbolic inputs by genuine values.

we audit uses of metamorphic testing to particular issue areas, and condense approaches that use metamorphic testing to improve other testing methods.

METAMORPHIC TESTING IS ALSO USED IN THE FOLLOWING DOMAINS:-

- Web Services and Applications
- Computer Graphics
- Simulation and Modeling
- Bioinformatics
- Components
- Numerical programs
- Compilers

Authors have explored their work in these domains and they proposed papers which are illustrated below :-

WEB SERVICES AND APPLICATIONS :-

Chan et al. [7], [8] proposed a metamorphic testing procedure for Service–Oriented Applications (SOA). Their strategy depends on the utilization of metamorphic services to typify the services under test, execute source and follow–up test cases and check their outcomes.

COMPUTER GRAPHICS :-

Mayer and Guderlei [9] looked at a few random images generation techniques for testing image handling programs. The study was performed on the usage of a image processing operator, in particular the Euclidean distance transform.

SIMULATION AND MODELING :-

Sim et al. [10] displayed an utilization of metamorphic testing for casting simulation exploiting the properties of the Medial Axis geometry capacity.

BIOINFORMATICS :-

Chen et al. [11] exhibited a few metamorphic relations for the detection of bugs in two open–source bioinformatics programs for gene regulatory network and short arrangement mapping.

COMPONENTS:-
Beydeda[12]proposed a self-testing method for commercial off the-shell components using metamorphic testing.

NUMERICAL PROGRAMS:-
Chen et.al[13] gives a case study on applications of metamorphic testing to programs implementing partial differential equations.

COMPILERS:-
Tao et al.[14] presented "equivalence preservation" metamorphic relations to test compilers.

COMPARISON OF PAPERS:-

Year	**Author's**	**Paper**	**Technique Used**	**Pros**	**Cons**
2004	T.Y. Chen	Metamorphic Testing and Its Applications	Metamorphic relation, successful test case, automated testing.	Check Result without an oracle and is very efficient.	Not only sufficient to detect faults but needed to combine with other testing methods.
2004	T.Y Chen	Metamorphic Testing and Beyond	Follow-up test cases, metamorphic testing, semi-proving, successful test case, test case selection strategy, testing oracle.	Efficient and reliable method.	Needs further research in other application domains.
2003	Gotlieb and Botella	Automated Metamorphic Testing	Constraint Logic Programming(CLP)	first attempt to generate automatically test data to reveal faults by the mean of Metamorphic Testing.	It is no so efficient to reveal the more faults.
2004	T.Y Chen	Meta Testing and Testing with special values"	Sine Function and Search Function.	It demonstrate the usefulness of metamorphic testing.	It needs further research in many application domains to detect more faults.
2011	S.Segura	Automated Metamorphic testing on the analyses of feature model.	Feature Models(FM)	More efficient and effective in detecting faults in a few seconds without the need of an oracle.	Anomalies are detected in the models such as redundancies or dead features .
2011	G.Batra and J.Sangupta	An efficient metamorphic testing technique using genetic algorithm.	Genetic Algorithm	Reveals Error quickly and efficiently.	Too Lengthy procedure and calculations.
2012	L. Chen, L. Cai, J. Liu, Z. Liu, S. Wei, and P. Liu	An optimized method for generating cases of metamorphic testing"	Black-Box Testing	Generate a small set of test cases with high detection rate.	Consumes more time.

2013	G. Dong, T. Guo, and P. Zhang	Security assurance with program path analysis and metamorphic testing	Symbolic Execution	It has ability to detect domain-specific faults.	Effectiveness is limited.
2014	Itti Hooda and Rajender Chhillar.	Study of Test Case Generation Techniques	Genetic Algorithm and Extended Finite State Machine(EFSM)	Requires minimum time, maximum fault coverage.	Further needs investigation to increase the software reliability.

III. CONCLUSION AND FUTURE SCOPE

we audit uses of metamorphic testing to particular issue areas, and condense approaches that utilizes metamorphic testing to upgrade other testing techniques. By perusing numerous papers identified with metamorphic testing presume that metamorphic testing is greatly improved than some other testing and till now it is connected to numerous areas. Metamorphic testing is extremely productive for distinguishing the bugs in the framework that are left undetected by ordinary testing, with this also the effectiveness of the framework gets expanded. As till now test cases are produced by utilizing random testing or by utilizing existing test suits as source test cases while applying metamorphic testing. Our future work is to generate the source test cases using software technique i.e. Flower pollination algorithm .

ACKNOWLEDGMENT

I would like to thank T.Y chen, A.Gotlieb and B.Botella, G.Batra, Sangupta, W.K chan, J.Mayer and Sergio segura for their helpful work in metamorphic testing.

REFERENCES

[1] A. Gotlieb and B. Botella, “Automated metamorphic testing,” inProceedings of the 27th Annual International Conference on ComputerSoftware and Applications, ser. COMPSAC ’03. Washington, DC,USA: IEEE Computer Society, 2003, pp. 34–. [Online].Available:http://dl.acm.org/citation.cfm?id=950785.950794

[2] T. Y. Chen, F.-C. Kuo, Y. Liu, and A. Tang, “Metamorphic testing and testing with special values,” in 5th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing, 2004, pp. 128–134.

[3] S. Segura, R. M. Hierons, D. Benavides, and A. Ruiz-Cort´es, “Automated metamorphic testing on the analyses of feature models,” Information and Software Technology, vol. 53, no. 3, pp. 245 – 258, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0950584910001904

[4] G. Batra and J. Sengupta, “An efficient metamorphic testing technique using genetic algorithm,” in Information Intelligence, Systems, Technology and Management, ser. Communications in Computer and Information Science, S. Dua, S. Sahni, and D. Goyal, Eds. Springer Berlin Heidelberg, 2011, vol. 141, pp. 180–188. [Online]. Available: http://dx.doi.org/10.1007/ 978-3-642-19423-8 19

[5] L. Chen, L. Cai, J. Liu, Z. Liu, S. Wei, and P. Liu, “An optimized method for generating cases of metamorphic testing,” in 6th International Conference on New Trends in Information Science and Service Science and Data Mining (ISSDM), 2012, Oct 2012, pp. 439–433G. Dong, T. Guo, and P. Zhang, “Security assurance with program path analysis and metamorphic testing,” in 4th IEEE International Conference on Software Engineering and Service Science (ICSESS),2013, May 2013, pp. 193–197.

[6] G. Dong, T. Guo, and P. Zhang, “Security assurance with program path analysis and metamorphic testing,” in 4th IEEE International Conference on Software Engineering and Service Science (ICSESS),2013, May 2013, pp. 193–197.

[7] W. K. Chan, S. C. Cheung, and K. R. P. Leung, “Towards a metamorphic testing methodology for service-oriented software applications,” in Fifth International Conference on Quality Software, 2005. (QSIC 2005), Sept 2005, pp. 470–476.

[8] W. K. Chan, S. C. Cheung, and K. R. P. H. Leung, “A metamorphic testing approach for online testing of service-oriented software applications.” International Journal of Web Services Research, vol. 4, no. 2, pp. 61–81, 2007. [Online]. Available: http: //dblp.uni-trier.de/db/journals/jwsr/jwsr4.html#ChanCL07.

[9] J. Mayer and R. Guderlei, “On random testing of image processing applications,” in Sixth International Conference on Quality Software, 2006. QSIC 2006, Oct 2006, pp. 85–92.

[10] K. Y. Sim, W. K. S. Pao, and C. Lin, “Metamorphic testing using geometric interrogation technique and its application,”in Proceedings of the 2nd International Conference of Electrical Engineering Electronics, Computer, Telecommunications, and Information Technology, 2005, pp. 91–95.

[11] T. Y. Chen, J. W. K. Ho, H. Liu, and X. Xie, “An innovative approach for testing bioinformatics programs using metamorphic testing,” BioMed Central Bioinformatics Journal, vol. 10, no. 1, p. 24,

2009. [Online]. Available: http://www.biomedcentral.com/1471-2105/10/24.

[12] S. Beydeda, "Self-metamorphic-testing components," in 30th Annual International Computer Software and Applications Conference, 2006. COMPSAC '06, vol. 2, Sept 2006, pp. 265–272.

[13] T. Y. Chen, J. Feng, and T. H. Tse, "Metamorphic testing of programs on partial differential equations: A case study," in Proceedings of the 26th International Computer Software and Applications Conference on Prolonging Software Life: Development and Redevelopment, ser. COMPSAC '02. Washington, DC, USA: IEEE Computer Society, 2002, pp.327333.[Online].Available:http://dl.acm.org/citation.cfm?id=645984.675903

[14] Q. Tao, W. Wu, C. Zhao, and W. Shen, "An automatic testing approach for compiler based on metamorphic testing technique,"in 17th Asia Pacific Software Engineering Conference (APSEC), 2010, Nov 2010, pp. 270–279.