ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/4174603

Iterative Metamorphic Testing

CONFERENCE PAPER · AUGUST 2005

DOI: 10.1109/COMPSAC.2005.93 · Source: IEEE Xplore

CITATIONS
13

READS
22

1 AUTHOR:

Peng Wu
Chinese Academy of Sciences
23 PUBLICATIONS **156** CITATIONS

SEE PROFILE

All in-text references underlined in blue are linked to publications on ResearchGate, letting you access and read them immediately.

Available from: Peng Wu
Retrieved on: 27 March 2016

Iterative Metamorphic Testing*

Peng Wu
Lab of Computer Science, Institute of Software, Chinese Academy of Sciences
Graduate School of the Chinese Academy of Sciences
Beijing 100080, China
E-mail: wp@ios.ac.cn

Abstract

An enhanced version of metamorphic testing, namely n-iterative metamorphic testing, is proposed to systematically exploit more information out of metamorphic tests by applying metamorphic relations in a chain style. A contrastive case study, conducted within an integrated testing environment MTest, shows that n-iterative metamorphic testing exceeds metamorphic testing and special case testing in terms of their fault detection capabilities. Another advantage of n-iterative metamorphic testing is its high efficiency in test case generation.

1. Introduction

Testing is a practical technique to validate the correctness of software. A successful test is the one that can detect faults in the software under test (SUT). A test that SUT has passed does not contribute to fault detection. Metamorphic testing was proposed to exploit information from such tests by examining multiple executions of SUT against some metamorphic relations[3]. A remarkable advantage of this method over other testing methods is that it does not require oracle, which is one of intractable limitations on software testing[7, 6].

Suppose SUT computes an n-nary function $f : D_1 \times \cdots \times D_n \rightarrow D_{out}$, which is also referred to as the specification that SUT must conform to. Let D denote its input domain, i.e. $D = D_1 \times \cdots \times D_n$. A metamorphic relation of the function f is defined as a pair $MR = (R, R_f)$ such that $R \subseteq \bigcup_{k\geq 2} D^k, R_f \subseteq \bigcup_{k\geq 2} (D \times D_{out})^k$ and for any $\bar{x}_1, \ldots, \bar{x}_k \in D$,

$$\text{If } (\bar{x}_1, \ldots, \bar{x}_k) \in R, ((\bar{x}_1, f(\bar{x}_1)), \ldots, (\bar{x}_k, f(\bar{x}_k)) \in R_f$$

A metamorphic relation MR is a necessary condition for the correctness of SUT. Let P be an implementation of the function f, and $P(\bar{x}_1), \ldots, P(\bar{x}_k)$ the outputs resulted from k inputs $\bar{x}_1, \ldots, \bar{x}_k \in D$ respectively. If P is correct, that is, conforming to its specification f, the following proposition should hold.

$$\text{If } (\bar{x}_1, \ldots, \bar{x}_k) \in R, ((\bar{x}_1, P(\bar{x}_1)), \ldots, (\bar{x}_k, P(\bar{x}_k)) \in R_f$$

Given a metamorphic relation $MR = (R, R_f)$ and a set of test cases $\{t_1, \ldots, t_{k-1}\}(k \geq 2)$ that have already been performed but without detecting any fault in the SUT, a fresh test case t_k can be derived in accordance with R. By examining these k executions against MR, metamorphic testing can not only detect the invalidity of SUT[4, 5, 14], but also identify its immunity from pre-specified faults in the absence of test oracles[7].

However, previous efforts just apply metamorphic relations directly and separately onto predefined test cases. Metamorphic relations, satisfied by the SUT, are also regarded as helpless as those passed test cases are. Therefore how to select metamorphic relations becomes a key issue for metamorphic testing[14].

We present in this paper an enhanced version of metamorphic testing, namely *n-iterative metamorphic testing* that can systematically exploit more information out of metamorphic tests by applying metamorphic relations in a chain style. Given a metamorphic relation sequence MR_1,.., MR_n, new test cases resulted from metamorphic relation $MR_i(1 \leq i < n)$ can be reused as source test cases for the next metamorphic relation MR_{i+1}. A contrastive case study shows that n-iterative metamorphic testing can practically improve the effectiveness of metamorphic testing. We have extended the integrated environment MTest[14] to perform iterative metamorphic tests automatically.

Related Work [6] presented a brief survey on metamorphic testing with potential research directions. [7] proposed to enhance fault based testing to address the oracle problem with metamorphic testing. [4, 5, 14] presented several

* This work was supported by the National Natural Science Foundation of China (under Grants 60223005 and 60421001) and the Chinese Academy of Sciences.

Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC'05)
0730-3157/05 $20.00 © 2005 IEEE

Authorized licensed use limited to: University of Wollongong. Downloaded on August 15,2010 at 11:26:52 UTC from IEEE Xplore. Restrictions apply.

meaningful case studies on metamorphic testing: [4] illustrated the applicability of metamorphic testing on solving partial differential equations, but with no comparison on the effectiveness of metamorphic testing against other testing methods; [5] demonstrated the use of metamorphic testing to detect faults in programs, which could not be detected by special test values; [14] gave a more practical and general consideration of special case testing and presented a more systematic evaluation and comparison with statistical data. n-iterative metamorphic testing can be regarded as a general way for test case generation with metamorphic relations.

The rest of the paper is organized as follows. Section 2 illustrates two algorithms of n-iterative metamorphic testing with different inputs. Section 3 describes the architecture and workflow of the revised integrated testing environment MTest. A case study with detailed experiment results are presented in Section 4. The paper is concluded with Section 5 where future work is also outlined.

2. n-Iterative Metamorphic Testing

This section will describe two algorithms of n-iterative metamorphic testing with different inputs: one works on a metamorphic relation sequence; while the other on a set of metamorphic relations.

Let $\sigma = MR_1, \ldots, MR_n$ be a metamorphic relation sequence, of which the length is $n(n \geq 0)$. For any $i \leq n$, $\sigma_i = MR_1, \ldots, MR_i$ is termed as a prefix of σ. Especially, $\sigma_n = \sigma$.

Figure 1 describes the workflow of IMT_q, an n-iterative metamorphic testing algorithm with a metamorphic relation sequence σ_n and a set of initial test cases $T_0 = \{t_1, \ldots, t_{k-1}\}$ $(k \geq 2)$.

Termination With the finite sequence σ_n of metamorphic relations, the algorithm IMT_q can terminate eventually at one of the following exits:

1. The SUT aborts on some test case at step 1 or 3(b)ii;
2. The outputs of SUT violate some metamorphic relation at step 3(b)iii;
3. All metamorphic relations have been run out at step 4.

Soundness The algorithm IMT_q endues the concept of the soundness with the sense that if the result of a test execution is fail then the SUT is sure to be not correct. In the former two cases above, IMT_q reports a failure due to certain exception in the SUT(1) or non-conformance of SUT to certain metamorphic relation(2); while in the last case, the result of IMT_q shows that no fault has been detected.

Algorithm $IMT_q(\sigma_n, T_0)$

1. Test SUT with source test cases $t_1, \ldots, t_{k-1}$ and record its results $SUT(t_1), \ldots, SUT(t_{k-1})$, respectively. If any source test case causes an unexpected exception resulting in the abnormal termination of SUT, e.g. bus error, segmentation fault etc., the test can be terminated with failure reported.
2. Let $T = \{t_1, \ldots, t_{k-1}\}$;
3. Apply metamorphic relations $MR_1, \ldots, MR_n$ one by one. For each metamorphic relation $MR_i = (R^i, R^i_f)$:
   (a) Let $T^i_f = \emptyset$;
   (b) For each fresh test case $t_k \notin T$, derived from source test cases in T in accordance with relation R^i,
      i. Test SUT with t_k;
      ii. If SUT terminates at t_k abnormally by certain unexpected exception, the test will be terminated with failure reported.
      iii. If SUT terminates normally with output $SUT(t_k)$, validate it according to R^i_f. If $R^i_f((t_1, SUT(t_1)), ..., (t_k, SUT(t_k)))$ does not hold, a failure will be reported; otherwise, $T^i_f := T^i_f \cup \{t_k\}$.
   (c) $T := T \cup T^i_f$.
4. If no failure is reported, SUT is deemed to pass all test cases.

Figure 1. n-iterative metamorphic testing with a metamorphic relation sequence σ_n and a test set T

Example To illustrate how IMT_q works, take a case study on sparse matrix multiplication, reported in [14], for an instance. In the case study, a C program that is a part of JASPA [11] is chosen as a reference implementation for sparse matrix multiplication. In one of its mutants, namely Mutant 4, a multiplication expression is replaced by its first operand.

One of the special test cases $SC_3 = (E, B)$, where E is an $n \times n$ identity matrix and B a random $n \times m$ $(n > 0, m > 0)$ matrix, cannot detect Mutant 4 itself as it does not cover the statement mutated. The metamorphic relation MR_2 exchanges some two rows of E, while MR_8 increases the values of the principal diagonal elements of E by 1.

Let MC^2_3 and MC^8_3 denote the resulted test cases by applying MR_2 and MR_8 onto SC_3, respectively. Neither MC^2_3 nor MC^8_3 can detect Mutant 4 for the same reason. But by applying MR_8 onto MC^2_3, that is, E is transformed first according to MR_2, of which the outcome is then trans-

Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC'05)
0730-3157/05 $20.00 © 2005 IEEE
Authorized licensed use limited to: University of Wollongong. Downloaded on August 15,2010 at 11:26:52 UTC from IEEE Xplore. Restrictions apply.

formed according to MR_8, Mutant 4 can be easily detected as the statement mutated make the computation result of Mutant 4 not satisfy the post-condition specified in MR_8.

From this example, it can be easily seen that iterative application of metamorphic relations can reach higher structure coverage and therefore increase the fault detection capability of metamorphic testing.

There may be only $m(m \leq n)$ different metamorphic relations in σ_n. IMT_q just considers one sequence(or string) of these m metamorphic relations. A straight strategy to extend it is to enumerate all sequences of length n on these m metamorphic relations, and then to call IMT_q in batches with these sequences. However, the time complexity of this strategy is intrinsically exponential because there are totally m^n different sequences available. Moreover, quite a number of test cases have to be derived repeatedly by those sequences that have common prefixes and then supplied to the SUT redundantly.

Figure 2 presents a practical on-the-fly algorithm IMT that works directly on a set of metamorphic relations $S = \{MR_1, \ldots, MR_m\}$. The basic idea is to derive metamorphic relation sequences dynamically during the test procedure. The main advantage of IMT is its efficiency of test case generation in the sense that redundant test cases mentioned above can be avoided.

Termination Similarly, IMT can terminate well because the number of rounds of iterations is bounded. During the test procedure, IMT may exit in one the following cases:

1. The SUT aborts on some test case at step 1 or 5b;
2. The outputs of SUT violate some metamorphic relation at step 5c;
3. All metamorphic relations have been run out at step 7.

Soundness IMT is sound as each fail verdict from IMT is always based on certain unexpected behavior of SUT. In detail, in the former two cases, IMT reports a failure due to certain exception in the SUT(1) or non-conformance of SUT to certain metamorphic relation(2); while in the lase case, the result of IMT shows that no fault has been detected.

With each metamorphic relation $MR_i \in S$, fresh test cases can be derived from only initial test cases in T_0 or from both initial test cases in T_0 and interim test cases that have been derived before. In this way, n-iterative metamorphic testing, no matter IMT_q or IMT, not only inherits the advantage of metamorphic testing, but also enhances its effectiveness in that

1. n-iterative metamorphic testing addresses the case when metamorphic relations can not help detect more faults on their own. It extends the workflow of metamorphic testing and provides a systematic way to derive various test cases by composing those metamorphic relations in a chain style.

2. Due to the combinatorial effect, far more test cases can be derived automatically without requiring manual efforts in n-iterative metamorphic testing than in metamorphic testing. Thus, n-iterative metamorphic testing can cover a wider part of the test domain in a manageable way.

IMT_q and IMT are both suitable for practical testing but with a bit different effect. When IMT terminates with a fail verdict, the metamorphic relation sequence that detect the fault can be recorded down for future regression testing with IMT_q.

Algorithm $IMT(T_0, S)$

1. Test SUT with source test cases $t_1, \ldots, t_{k-1}$ and record its results $SUT(t_1), \ldots, SUT(t_{k-1})$, respectively. f any source test case causes an unexpected exception resulting in the abnormal termination of SUT, the test can be terminated with failure reported.
2. Let $T = \{t_1, \ldots, t_{k-1}\}$;
3. Repeat step 4 to 6 n times;
4. Let $T_f^i = \emptyset$;
5. For each metamorphic relation $MR_i = (R^i, R_f^i)$ in S and each fresh test case $t_k \notin T$ derived from source test cases in T in accordance with relation R^i:
   (a) Test SUT with t_k;
   (b) If SUT terminates at t_k abnormally by certain unexpected exception, the test will be terminated with failure reported.
   (c) If SUT terminates normally with output $SUT(t_k)$, validate it according to R_f^i. If $R_f^i((t_1, SUT(t_1)), ..., (t_k, SUT(t_k)))$ does not hold, a failure will be reported; otherwise, $T_f^i := T_f^i \cup \{t_k\}$.
6. $T := T \cup T_f^i$.
7. If no failure is reported, SUT is deemed to pass all test cases.

Figure 2. n-iterative on-the-fly metamorphic testing

Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC'05)
0730-3157/05 $20.00 © 2005 **IEEE**
Authorized licensed use limited to: University of Wollongong. Downloaded on August 15,2010 at 11:26:52 UTC from IEEE Xplore. Restrictions apply.

3. Implementation

Based on previous work[14], we extend the integrated environment MTest to be able to automatically perform iterative metamorphic testing with special and random inputs as initial test cases. The architecture of the test environment is shown in Figure 3. Given a SUT, MTest customizes a metamorphic tester with two configuration files: one supplies special cases and corresponding oracles of SUT, while the other supplies the interface definition and metamorphic relations of SUT.

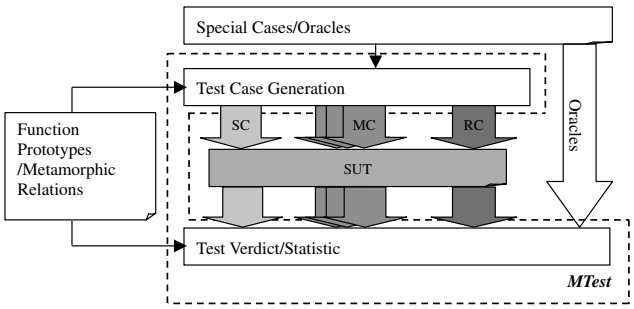

Figure 3. Iterative Metamorphic Test Architecture

The metamorphic tester consists of two parts: Test Case Generation and Test Verdict/Statistic. The former is to derive test cases and call SUT with them thereafter, while the latter is to collect outputs from SUT and then determine whether SUT passes test cases according to pre-defined test oracles or metamorphic relations. A test case may be a special test case(denoted as SC); or a random test case(denoted as RC); or resulted from transforming prevenient test cases in accordance with certain metamorphic relation(denoted as MC).

The tester can carry out iteratively as many rounds of test campaigns as configured by the user. In each round, the variations of prevenient test cases in accordance with each metamorphic relation are applied to SUT for more extensive tests. Special test cases, if supplied, can be used as initial test cases; otherwise, random test cases are to be used. Finally the tester reports verdicts of SUT and counts the number of test cases that SUT fails in each round of test.

4. Case Study

To compare the effectiveness between n-iterative metamorphic testing and other testing methods, especially metamorphic testing, we continue our case study[14] on spare matrix multiplication with revised measurements. This section will briefly introduce the measurements and illustrate our experiment results on n-iterative metamorphic test campaigns that have been performed automatically with MTest. For detailed information about the reference implementation, as well as it mutants and metamorphic relations, please refer to [14].

4.1. Measurement

In our research, mutation analysis technique[10, 2] are employed to evaluate and compare the effectiveness of testing methods, which can then be measured quantitatively at two levels in different granularities: one is by counting the number of mutants that could be detected; and the other by calculating how many of its test cases are able to detect a particular mutant. Correspondingly two measurements are considered herein: mutation score and fault detection ratio.

In mutation analysis, the quality of a test set is measured by mutation score, which is the percentage of mutants detected, that is

$$MT(T) = \frac{M_k}{M_t - M_q}$$

where T is the test set; M_k the number of mutants detected by T, M_t the total number of mutants, and M_q the number of equivalent mutants that cannot be detected by any set of test data.

Mutation score is also appropriate for assessing the effectiveness of a testing method. Regarding test sets resulted from different testing methods, higher mutation score implies higher quality of the corresponding test set. Therefore, mutation score can serve as a macroscopical measurement in the sense that it does not matter which mutant is detected or not. The definition above can be reused for this task except that T stands for a testing method in the sense that multiple test sets derived by the testing method are evaluated and results are averaged as the measurement for the testing method.

On the other hand, a microscopical measurement is desirable to analyze to what extent a testing method can detect a fault in SUT, in the sense that each mutant is considered separately. A simple way is to calculate the fault detection ratio FD, the percentage of test cases that could detect certain mutant M, that is

$$FD(M, T) = \frac{N_f}{N_t - N_e}$$

Where N_f is the number of times SUT fails, N_t the number of tests, and N_e the number of infeasible tests. For our case study, where only binary metamorphic relations are applied,

$$N_t = N_c \times \sum_{i=1}^{n} m^i$$

Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC'05)
0730-3157/05 $20.00 © 2005 **IEEE**
Authorized licensed use limited to: University of Wollongong. Downloaded on August 15,2010 at 11:26:52 UTC from IEEE Xplore. Restrictions apply.

where $N_c$is the number of initial test cases, m the number of metamorphic relations and n the number of rounds of iterations, while N_e is the total number of times that source test cases could not satisfy the requirement of a metamorphic relation. It can be easily seen that the number of candidate test cases N_t in n-iterative metamorphic testing is far greater than the one in metamorphic testing, since each initial test case may result in a corresponding test case for each metamorphic relation in each round of iteration.

4.2. Experiment Results

As described in [14], five mutants are designed for the sparse matrix multiplication function based on two types of mutation operators: SDL (Statement Deletion) and AOR (Arithmetic Operator Replacement), which were defined in a mutation-testing environment Mothra[9, 13]. Nine metamorphic relations are applied in the case study.

Table 1 and 2 illustrate respectively the experiment results of n-iterative metamorphic testing on Mutant 4 and Mutant 5, which cannot be detected by special test cases with any metamorphic relation. Column 1 lists the initial test cases used for subsequent metamorphic tests. Column 2 to 4 show the metamorphic relation sequences that can detect these two mutants within no more than three rounds of iterations, respectively. $i - j - \cdots$ means that MR_i is applied on the corresponding initial test case(at Column 1), followed by MR_j on the resulted test case and so forth. $j_1/\cdots/j_k$ means that any of these k metamorphic relations can be used in that step.

Mutant 4	1-step	2-step	3-step
SC_1	-	-	8-2-8
SC_2	-	-	-
SC_3	-	2-8	3/4/5/6/7/8/9-2-8 2-3/4/5/6/7/9-8
SC_4	-	-	3-9-1/5/7 3-1-8 1-2-8 7/9-3-9 3-5/7/9-9
SC_5	-	-	-
SC_6	-	-	-
SC_7	-	-	-
SC_8	-	-	-
N_{total}	70	614	5389
FD	0	0.1627%	0.4454%

Table 1. Iterative Metamorphic Testing Mutant 4

Mutant 5	1-step	2-step	3-step
SC_1	-	-	-
SC_2	-	-	9-3-9
SC_3	-	-	2-8-1/4/6 6/8-2-8 2-4/6/8-8 2-1-9 1-3-9
SC_4	-	3-9	3-2-9 2/4/5/6/7/8/9-3-9 3-4/5/6/7/8-9
SC_5	-	-	-
SC_6	-	-	-
SC_7	-	-	-
SC_8	-	-	-
N_{total}	70	614	5389
FD	0	0.1627%	0.4454%

Table 2. Iterative Metamorphic Testing Mutant 5

It can be easily seen that these two mutants can be detected through metamorphic relation sequences, of which the length is as short as two. Therefore, the mutation score for iterative metamorphic testing is 1 and larger than special case testing and metamorphic testing with special test cases, which are both 0.6.

Only five mutants may not ascertain the effectiveness of n-iterative metamorphic testing. We introduce a mutation tool Proteum[8] in our case study for automated mutant generation so that the effectiveness of n-iterative metamorphic testing can be evaluated in a more general sense.

Proteum is a testing tool that supports mutation analysis criterion. With the help of Proteum, totally 1419 mutants are generated, among which 99 mutants are equivalent to the reference implementation. Table 3 illustrates the corresponding experiment results.

	#Detected	Mutation Score
Special Case Testing	1176	0.8909
1-Iterative Metamorphic Testing	1180	0.8939
2-Iterative Metamorphic Testing	1316	0.9970
3-Iterative Metamorphic Testing	1316	0.9970
4-Iterative Metamorphic Testing	1316	0.9970
5-Iterative Metamorphic Testing	1316	0.9970

Table 3. Special Case Testing vs. Iterative Metamorphic Testing

Proceedings of the 29th Annual International Computer Software and Applications Conference (COMPSAC'05)
0730-3157/05 $20.00 © 2005 **IEEE**
Authorized licensed use limited to: University of Wollongong. Downloaded on August 15,2010 at 11:26:52 UTC from IEEE Xplore. Restrictions apply.

It can be easily seen that all but four non-equivalent mutants can be detected by n-iterative metamorphic testing with $n > 1$. n-iterative metamorphic testing exceeds metamorphic testing and special case testing by a bit more than ten percent in terms of their mutation scores. Those missed mutants, namely Mutant 722, 748, 754 and 766, are generated by applying a trap operator TRAP_ON_NEGATIVE on the elements of input matrixes. The operator will kill the current process if certain element is negative. In our case study, all input matrixes contain only positive integers, which leaves those four non-equivalent mutants unperceived.

5. Conclusion

We have proposed an enhanced version of metamorphic testing, namely n-iterative metamorphic testing that can address the case when metamorphic relations can not help detect any fault in the SUT on their own. The main advantage of this testing method lies in its high efficiency on test case generation with better fault detection capability than metamorphic testing and special case testing. A contrastive case study ascertains its effectiveness with the help of mutation test tool Proteum.

It is worth noting that n-iterative metamorphic testing is not limited to numerical problems. Metamorphic relations can be identified in various non-numerical areas, such as document text extracting, natural language comprehension etc. [15] also illustrates a wide range of applications of metamorphic testing. Thus n-iterative testing can be naturally applied into these areas.

From the case study, it can be concluded that more rounds of iterations may not result in more mutants detected. The concrete relationship between the number of rounds of iterations and the testing effectiveness is still left open.

On the other hand, we would also like to investigate the way to integrate n-iterative metamorphic testing with other testing methods and to embed it into practical software development process[12, 1] as a flexible means for software quality assurance.

References

[1] K. Beck. *Extreme Programming Explained*. Addison-Wesley, 2000.

[2] T. A. Budd. Mutation analysis: Ideas, examples, problems and prospects. In C. B. and R. S., editors, *Proceedings of the Summer School on Computer Program Testing*, pages 129–148, Amsterdam, 1981. North-Holland.

[3] T. Y. Chen, S. C. Cheung, and S. M. Yiu. Metamorphic testing: A new approach for generating next test cases. Technical Report Technical Report HKUST-CS98-01, Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong, 1998.

[4] 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 Annual International Computer Software and Applications Conference*, Oxford, England, August 26 - 29 2002.

[5] T. Y. Chen, F.-C. Kuo, Y. Liu, and A. Tang. Metamorphic testing and testing with special values. In *Proceedings of the 5th ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD2004)*, Beijing, China, June 30 - July 2 2004.

[6] T. Y. Chen, F.-C. Kuo, T. H. Tse, and Z. Zhou. Metamorphic testing and beyond. In *Proceedings of the International Workshop on Software Technology and Engineering Practice*, 2003.

[7] T. Y. Chen, T. H. Tse, and Z. Q. Zhou. Fault-based testing in the absence of an oracle. In *Proceedings of the 25th Annual International Computer Software and Application Conference*, 2001.

[8] M. E. Delamaro and J. Maldonado. Proteum-a tool for the assessment of test adequacy for c programs: User's guide. Technical Report SERC-TR-168-P, Software Engineering Research Center, April 1996.

[9] R. A. DeMillo, D. S. Guindi, W. M. McCracken, A. J. Offutt, and K. N. King. An extended overview of the mothra software testing environment. In *Proceedings of the 2nd Workshop on Software Testing, Verification, and Analysis*, pages 142–151, Banff, Canada, July 19-21 1988.

[10] R. A. DeMillo, R. J. Lipton, and F. G. Sayward. Hints on test data selection: Help for the practicing programmer. *Computer*, 11(4):34–41, 1978.

[11] http://www.cse.clrc.ac.uk/arc/JASPA/. CSE - JASPA: a sparse matrix multiplication benchmark for JAVA/F90/C.

[12] P. Kruchten. *The Rational Unified Process: An Introduction*. Addison-Wesley, 2nd edition, 2000.

[13] R. H. Untch, A. J. Offutt, and M. J. Harrold. Mutation analysis using mutant schemata. In *Proceedings of the 1993 ACM SIGSOFT international symposium on Software testing and analysis*, pages 139–148. ACM Press, 1993.

[14] P. Wu, X. Shi, J. Tang, H. Lin, and T. Y. Chen. Metamorphic testing and special case testing: A case study. Accepted by Journal of Software, 2004.

[15] Z. Q. Zhou, D. H. Huang, T. H. Tse, Z. Yang, H. Huang, and T. Y. Chen. Metamorphic testing and its applications. In *Proceedings of the 8th International Symposium on Future Software Technology (ISFST 2004)*, Xian, China, October 20 - 22 2004.

Authorized licensed use limited to: University of Wollongong. Downloaded on August 15,2010 at 11:26:52 UTC from IEEE Xplore. Restrictions apply.