Teaching an end-user testing methodology

Liu, Huai; Kuo, Fei Ching; Chen, Tsong Yueh

Liu, H., Kuo, F. C., & Chen, T. Y. (2010). Teaching an end-user testing methodology. Proceedings of 23rd International IEEE Conference on Software Engineering Education and Training, CSEE, 81–88. https://doi.org/10.1109/CSEET.2010.28
Document Version: Accepted Manuscript

Published Version: https://doi.org/10.1109/CSEET.2010.28

Teaching an End-User Testing Methodology

Huai Liu*, Fei-Ching Kuo, and Tsong Yueh Chen
Centre for Software Analysis and Testing, Swinburne University of Technology, Australia
{hliu, dkuo, tychen}@swin.edu.au

Abstract

One important focus of software engineering is how to develop quality software. Software testing is the main approach to the software quality assurance. Nowadays, more and more end-users write the program on their own but lack formal trainings on how to test their programs, and hence cannot guarantee the quality of their own software. Metamorphic testing is a simple, automatable, and cost-effective testing methodology. It is particularly suitable for end-users to test their own programs, because it does not demand the user to have great knowledge of software testing but knowledge of the program under development. In this paper, we report our experience in teaching metamorphic testing to various groups of students at Swinburne University of Technology, Melbourne, Australia. Our work not only enhances the teaching of software testing, but also fosters the training of end-user programmers.

1. Introduction

Software is extensively used all over the world, and has a profound impact on our life. One main challenge of software engineering is to build software with high quality within limited resources [7, 12]. Software testing, a major approach in assuring the software quality, can occupy half of resources during software development [14]. As a result, software testing becomes an important topic to be covered in the teaching program of software engineering [5, 9].

Nowadays, with the support of programming tools, many end-users are able to program the software. Unfortunately, most end-users do not have formal trainings on testing, and their programs may have serious defects [1, 4]. Therefore, it is urgent to teach some simple yet effective testing techniques to end-user programmers.

Recently, Chen *et al.* [2] proposed a novel testing methodology, namely *metamorphic testing* (MT). In MT, a tester first identifies some *metamorphic relations* (MRs) from the software specification, then generates test cases based on MRs, and finally verifies test results against MRs. Applying MT does not require much specific knowledge of software testing but domain knowledge of the program under development, and it can be easily automated. Some previous studies have also shown that MT is very effective in revealing program faults [2, 6]. Therefore, MT is simple and effective enough to be a suitable testing technique for end-user programmers [4]. We have taught MT as an end-user testing methodology at Swinburne University of Technology, Melbourne, Australia in 2005, 2007, and 2008. This paper reports our teaching experience.

*Corresponding author.

2. Challenges of teaching software testing

The "Software Testing and Reliability" subject has been taught at Swinburne University of Technology for about 10 years. The class was composed of both undergraduates (in Computer Science and Software Engineering) and postgraduates (in Information Technology). In the teaching of software testing techniques, we were confronted with the following challenges.

1. **Limited teaching hours**. The duration of the subject is only 12 weeks. In each week, there is a 2-hour lecture accompanied by a 1-hour tutorial. The curriculum must be carefully designed in order that students could gain an in-depth understanding of software testing within limited time.
2. **Lack of supporting tools**. Many software testing techniques, such as coverage testing [16], domain testing [15], and category-partition method [13], involve complex partitioning and test data generation processes. Though some automated tools are available to support these processes, they are either expensive or non-user-friendly.
3. **Different backgrounds and career orientations of students**. The subject targets both undergraduates and postgraduates. Students are from different backgrounds, and have different career targets. Some students may have worked as a programmer or even a tester, while others may have little experience in software engineering. Some students may want to become a professional software engineer after graduation, while others may just want to have some basic knowledge of software engineering. Since all IT students can become a programmer in the future (professional or end-user), it is important to ensure that our graduates can apply the learned techniques to their software products after completing the subject.

To meet these challenges, we attempted to teach students a testing method that is conceptually simple, easily used, automatable, cost-effective in revealing program faults, and always applicable even without the test *oracle* (that is, a systematic mechanism to verify the correctness of the test result given any program input).

3. Metamorphic testing and its application to end-user engineering

Software testing consists of activities of defining test objectives, designing and generating *test cases*, executing the program with test cases, and verifying test results against the test oracle. When a test oracle does not exist, it is meaningless to conduct testing as we cannot check whether the software passes the test or not. Such a problem is called the *oracle problem*. Chen *et al.* [2] proposed a novel testing methodology, namely *metamorphic testing* (MT), to alleviate the oracle problem.

Based on the domain knowledge of the software under test, testers first identify some necessary properties from the software specification. These properties are then used to derive expected relations (called *metamorphic relations*, abbreviated as MRs) among multiple inputs and outputs. While some traditional testing methods (such as random testing [11]) are used to generate one *source test case*, MRs are used to convert this source test case to some *follow-up test cases*. Both the source test case and follow-up test cases are executed on the program under test. The test outputs are verified by MRs, instead of the oracle. If the outputs of source and follow-up test cases do not satisfy their corresponding MRs, a failure is said to be revealed.

Apart from providing a test result verification mechanism alternative to the oracle, MT has many other advantages. First, the rationale behind MT is very simple, and it does not require

the tester to have much in-depth knowledge of software testing. Therefore, we can well utilize 12 weeks to teach students in MT and other testing principles. Second, MT does not require the support of existing testing tools, and its test case generation and test result verification processes can be easily automated [8, 10]. Hence, we can teach students the automation of MT in a short period. Third, MT is very cost-effective in detecting software faults [2, 6]. As a result, it can greatly improve the quality of the software product that our students may develop (though we do not know what our students will be in the future).

MT has been advocated to be particularly useful for end-user testing, as (1) "end-user programmers have the domain knowledge to identify MRs" and (2) "end-user programmers can distinguish good MRs based on program structures" [4]. For example, researchers from the communities of bioinformatics [3] have used MT to test their own programs based on their domain knowledge. We have taught MT as an end-user testing methodology to various groups of students in 2005, 2007, and 2008. Our teaching approach is introduced in the following section.

4. Our teaching approach

To teach MT, we utilized the teaching approach "lectures plus tutorials plus paper reading plus assignments".

The lectures introduced the basic concepts and principles of various software testing methods, including MT. Also presented were the concept of oracle problem and how MT alleviates the oracle problem. However, the details of how to implement MT was not covered in the lectures. A 1-hour tutorial was allocated to discuss such details. In this tutorial, tutors illustrated how to identify MRs, how to generate source and follow-up test cases, and how to verify test results based on MRs.

Two assignments were designed for students to learn MT. In Assignment 1, two papers [2, 4] were given to students for an in-depth understanding of MT, and a small project for practising MT. Based on the knowledge and experience of MT obtained in Assignment 1, students were required in Assignment 2 to develop a test driver that automates the MT process. For each assignment, one tutorial was assigned before the submission for tutors and students to discuss the assignments requirements; while another tutorial after the marking was used to provide feedback on the students' work, where the tutors corrected the common mistakes student had committed in order that students could avoid the same mistakes in the real-life practice.

We expected that through this teaching approach, students not only obtained an in-depth understanding of MT technique, but also learned how MT can alleviate a fundamental problem, namely oracle problem, in software testing.

5. The assignments

Our students were given two assignments from which they learned how to use the MT technique. Table 1 summarized the assignments in 2005, 2007, and 2008. Generally, Assignment 1 included two tasks, paper reading and a small project for using MT; while in Assignment 2, students were required to develop a test driver to automate MT. In 2005, both assignments were about Microsoft Excel spreadsheets; in 2007, both assignments were about scientific programs; while in 2008, Assignment 1 was about a scientific program, and Assignment 2 was about spreadsheets. Assignments of different years have similar degrees of challenge. Their design and motivation are explained in Section 5.1 and 5.2.

Table 1. Assignments in 2005, 2007, and 2008

Year	Assignment 1	Assignment 2
2005	(1) Paper reading; (2) A small project on an Excel spreadsheet	Development of an MT test driver on 2 Excel spreadsheets
2007	(1) Paper reading; (2) A small project on a scientific program	Development of an MT test driver on a scientific program
2008	(1) Paper reading; (2) A small project on a scientific program	Development of an MT test driver on 2 Excel spreadsheets

5.1. Design and motivation of Assignment 1

In Assignment 1, we first gave students two papers to read. Paper A [2] is the first article introducing MT. By reading Paper A, students were expected to understand the basic concepts and principles of MT. Paper B [4] advocated MT as an end-user testing methodology. By reading Paper B, students were expected to know the concept of end-user software engineering and why MT is particularly suitable for end-user testing.

Paper reading was accompanied by the exercise of question and answer (Q&A). In 2005, Q&A exercise required students to identify 2 problems that may involve end-user programming, and to define at least 5 MRs for each problem.

In 2007 and 2008, Q&A exercise required students to identify one problem (instead of two) for end-user programming and to identify at least 5 MRs for the problem. In addition, we prepared two more questions based on Paper A to further examine students' understanding of MT. One question is about the test case generation in MT, while the other is about the test result verification.

Apart from paper reading and Q&A exercise, students were asked to practise MT in a small project. Through this project, students were expected to understand how MT is conducted. The project was designed as follows.

In 2005, we gave students a specification, based on which students were required to work out a Microsoft Excel spreadsheet and then test the spreadsheet by MT. We asked students to identify at least 5 MRs, and to design and generate test cases based on these MRs. Students were required to manually apply these test cases to the spreadsheet, and then to verify the test results against MRs.

In 2007 and 2008, instead of asking students to create the software based on a specification, we gave them an executable program that they could test directly. This program reads in a file containing some decimal numbers, and outputs some statistical data for these numbers (such as mean value and standard deviation). We also gave students an input file to be used as the source test case for MT. Students were required to identify at least 2 MRs for each statistical function, use these MRs to manually generate follow-up test cases, execute all test cases, and verify test results.

5.2. Design and motivation of Assignment 2

In Assignment 2, we asked students to develop a test driver to automate MT. It was expected that through Assignment 2, students could appreciate the efficiency of automatic testing and understand the implementation of MT.

In 2005 and 2008, we gave students two Microsoft Excel spreadsheets and two well-defined MRs. Students were required to develop a test driver that automatically generates, executes, and

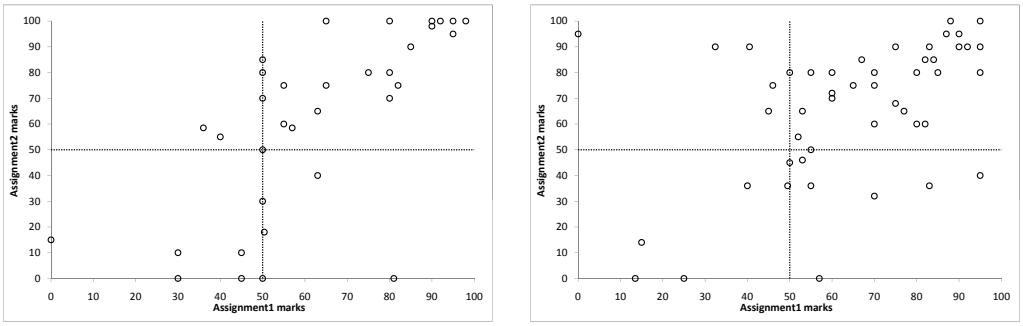
verifies test cases for the given spreadsheets and MRs.

In 2007, we gave students an executable program that constructs a simple undirected graph. We also gave students two MRs and eight source test cases, based on which students were required to develop a test driver that automatically generates follow-up test cases, executes both source and follow-up test cases, and verifies test results.

6. Discussion and recommendations

Based on students' performance on assignments and our close interaction with students, we have made the following observations.

- *Most students understood the concepts and principles of MT and learned how to use MT.* Fig. 1 shows the marks of assignments, where x- and y-axis denote the marks for Assignments 1 and 2, respectively. Among all 35 undergraduate students (Fig. 1(a)), there were 28, 25, and 23 students who passed (whose marks were not smaller than 50) Assignment 1, Assignment 2, and both assignments, respectively. Among all 53 postgraduate students (Fig. 1(b), where nine points are displayed as four points due to the overlapping of some students' scores), there were 42, 41, and 35 students who passed Assignment 1, Assignment 2, and both assignments, respectively. In summary, the passing rates for Assignment 1, Assignment 2, and both assignments are 79.5%, 75%, and 65.9%, respectively. Such high passing rates indicates that most students understood how MT works and thus were able to apply MT in testing software.



(a) Marks of undergraduate students (b) Marks of postgraduate students

Figure 1. Assignments marks in 2005, 2007 and 2008

- *Most students were able to identify correct MRs based on our guidance.* In Assignment 1, we required students to identify MRs for Excel and scientific programs. These programs are very simple, and we have given students some statistics and probability lessons. Based on the lessons, most of our students could correctly identify MRs for these programs.
- *Different students identified different MRs that target different faults.* Although most students could identify valid MRs, different MRs have different effectiveness in revealing different faults. For example, in Assignment 1 of 2007, students were required to identify at least 2 MRs for the function that calculates the maximal value from a set of input data. The correct code for such a function is given in Fig. 2(a), and the program we gave to students is shown in Fig. 2(b). The following summarizes 8 MRs identified by all 35 students. Suppose that there is a set of real numbers, $\mathbf{X} = \{x_1, x_2, \cdots, x_n\}$, and the maximal value in $\mathbf{X}$ is m.

- MR(i): Suppose that there is another set $\mathbf{X}' = \mathbf{X} \cup \{a\}$, where $a > m$, and the maximal value in $\mathbf{X}'$ is m'. We should have the relation $m' > m$.
- MR(ii): Suppose that there is another set $\mathbf{X}' = \mathbf{X} \cup \{b\}$, where $b < m$, and the maximal value in $\mathbf{X}'$ is m'. We should have the relation $m' = m$.
- MR(iii): Suppose that there is another set $\mathbf{X}'$ that is constructed by deleting all elements that are equal to m from $\mathbf{X}$, and the maximal value in $\mathbf{X}'$ is m'. We should have the relation $m' < m$.
- MR(iv): Suppose that there is another set $\mathbf{X}' = \{0.0 - x_1, 0.0 - x_2, \cdots 0.0 - x_n\}$, and the maximal value in $\mathbf{X}'$ is m'. We should have the relation $m' = c$, where c is the minimal value (which is one program output) in $\mathbf{X}$.
- MR(v): Suppose that there is another set $\mathbf{X}'$ that is constructed by changing the order of the data in $\mathbf{X}$, and the maximal value in $\mathbf{X}'$ is m'. We should have the relation $m' = m$.
- MR(vi): Suppose that there is another set $\mathbf{X}' = \{d * x_1, d * x_2, \cdots d * x_n\}$, where $d > 0$, and the maximal value in $\mathbf{X}'$ is m'. We should have the relation $m' = d * m$.
- MR(vii): Suppose that there is another set $\mathbf{X}' = \mathbf{X} \cup \{e\}$, where $e = m$, and the maximal value in $\mathbf{X}'$ is m'. We should have the relation $m' = m$.
- MR($viii$): Suppose that there is another set $\mathbf{D}' = \{f + x_1, f + x_2, \cdots f + x_n\}$, and the maximal value in $\mathbf{X}'$ is m'. We should have the relation $m' = f + m$.

```
1) max = 0.0 – DBL_MAX;
2) for(i = 0; i < number; i++)
3)     if(max < data[i]) max = data[i];
```

(a) Correct version

```
1) max = 0.0;   // Fault.
2) for(i = 0; i < number; i++)
3)     if(max < data[i]) max = data[i];
```

(b) Incorrect version in the assignment

```
1) max = 0.0 – DBL_MAX;
2) for(i = 1; i < number; i++)    // Fault.
3)     if(max < data[i]) max = data[i];
```

(c) Another incorrect version

Figure 2. Code to calculate maximal value

Among all 8 MRs, MR(i) is the most popular one (proposed by 20 of 35 students), followed by MR(ii) (7 students), MR(iii) (7 students), MR(iv) (6 students), MR(v) (4 students), MR(vi) (2 students), MR(vii) (1 student), and MR($viii$) (1 student). We have given students a file, which contained 10,000 positive decimal numbers, as a source test case. Given such a source test case, MRs (i), (ii), (iii), (v), (vi), and (vii) could not detect the fault shown in Fig. 2(b). MR(iv) will definitely reveal the fault, because after negating all values in the input file, all data will become negative and the output of the code in Fig. 2(b) will be 0, which violates MR(iv). MR($viii$) may detect the fault, if $f < 0.0 - m$. It should be noted that different faults may be sensitive to different MRs. For example, let us look at the fault shown in Fig. 2(c). For such a fault, MR(v) may be more effective than other seven MRs. Such an observation suggests that various MRs are needed to verify the correctness of the program output, and knowing the structure of the program may help identify various MRs for different types of faults [4]. We showed students the code in Fig. 2(c) and discussed this issue with students in a tutorial after the marking of Assignment 1. After the discussion, many students were able to find MRs that are effective in detecting different faults (such as the faults shown in Figures 2(b) and 2(c)). They also understood that end-user programmers have a full picture of the program structures, and thus may be able to identify MRs that are effective in detecting faults of their programs.

- *The majority of students were able to automate MT without the supporting tool.* In Assignment 2, students were required to develop a test driver that automates the MT process. In 2005, 18 of 35 students could implement both MRs correctly, 8 students could only implement one MR, and 9 students could not automate MT at all. In 2007, 26 of 35 students could automate MT based on both MRs, while other 9 students could not. In 2008, 9 of 18 students could implement both MRs correctly, 2 students could only implement one MR, and 7 students could not automate MT at all. In brief, the majority of students have gained the ability to automate MT.
- *The test drivers developed by different students have different failure-detection effectiveness.* Although the majority of students could develop an MT test driver in Assignment 2, their test drivers have different effectiveness in detecting failures. For example, in Assignment 2 of 2007, we required students to develop a test driver that automates MT based on two MRs and eight source test cases. In the lectures and tutorials, we have explicitly taught students that for one MR, one source test case is not necessarily associated with only one follow-up test case. However, most students only generated one follow-up test case based on one source test case for one MR, that is, their test drivers could only generate 16 follow-up test cases (8 source test cases $\times$ 2 MRs). 3 of 35 students developed MT test drivers that can generate a large number of follow-up test cases, and they could detect more different failures than other students. We have discussed such an observation with students in a tutorial after the marking of Assignment 2, and emphasized that MT can be easily automated to generate a large number of test cases to reveal many failures.

From the 3-year teaching experience, we gave the following recommendations for teaching MT. First, it is important for students to understand that domain knowledge is required to define good MRs and implement MT. Given such a short teaching and learning period (12 weeks), we recommend that in the teaching of MT, we should choose the software that most students are familiar with for them to practise the MT process. Second, it is important for students to learn that different faults are sensitive to different MRs, and we recommend that after discussing this issue with students, we should direct students to develop a list of diverse MRs that could cover various faults. Third, it is important for student to practise the identification of MRs, because such practice would help students improve their testing abilities. Finally, it is important for students to practise test automation, so we recommend that we should encourage students to generate as many test cases as possible by MT automation for increasing failure detection capabilities and for cost-effective testing.

7. Conclusion

In this paper, we reported our 3-year experience in teaching an end-user testing methodology under three constraints (limited time of teaching, lack of supporting tools, and various students' backgrounds and career orientations). Metamorphic testing (MT) is a simple yet effective testing method. It is a suitable testing technique for end-user engineering. The process of MT can be automated easily without existing supporting tools. Hence, students can learn MT and its automation in a short period of time. No matter what background students have and what students plan to do in the future, if one day they develop their own software, learning MT can be beneficial for them to effectively detect failures of their own software.

In these three years, we chose the teaching approach "lectures plus tutorials plus paper reading plus assignments" for teaching MT. With our careful design of curriculum, students have learned the process and automation of MT. They learned that different faults may be sensitive to different MRs, and hence it is essential to generate a list of diverse MRs for testing. They also understood that end-user programmers have a full picture of the program structures and thus may be able to identify MRs that are effective in revealing faults of their programs. Students also appreciated that MT can be easily automated to generate a large amount of test cases based on a few source test cases and several MRs.

Our 3-year teaching experience showed that "lectures plus tutorials plus paper reading plus assignments" is a good approach to teaching MT (an end-user testing methodology). The experience reported in the paper is also valuable for the training of end-user testing.

8. Acknowledgment

The authors would like to thank Dehao Huang and Xiaoyuan Xie for their contributions to the teaching of the subject in 2005 and 2008, respectively. They are also grateful to the students who were involved in this study. This project was supported by an Australia Research Council grant (DP0771733).

References

[1] M. Burnett, C. Cook, and G. Rothermal. End-user software engineering. *Communications of the ACM*, 47(9):53–58, 2004.

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

[3] T. Y. Chen, J. W. K. Ho, H. Liu, and X. Xie. An innovative approach for testing bioinformatics programs using metamorphic testing. *BMC Bioinformatics*, 10:24, 2009.

[4] T. Y. Chen, F.-C. Kuo, and Z. Q. Zhou. An effective testing method for end-user programmers. In *Proceedings of the 1st Workshop on End-User Software Engineering (WEUSE)*, pages 1–5, 2005.

[5] T. Y. Chen and P.-L. Poon. Experience with teaching black-box testing in a computer science/software engineering curriculum. *IEEE Transactions on Education*, 47(1):42–50, 2004.

[6] T. Y. Chen, T. H. Tse, and Z. Q. Zhou. Fault-based testing without the need of oracles. *Information and Software Technology*, 45(1):1–9, 2003.

[7] E. W. Dijkstra. The end of computing science? *Communications of the ACM*, 44(3):92, 2001.

[8] A. Gotlieb and B. Botella. Automated metamorphic testing. In *Proceedings of the 27th Annual International Computer Software and Applications Conference (COMPSAC 2003)*, pages 34–40, 2003.

[9] E. L. Jones and C. L. Chatmon. A perspective on teaching software testing. *Journal of Computing Sciences in Colleges*, 16(3):92–100, 2001.

[10] C. Murphy, K. Shen, and G. E. Kaiser. Automatic system testing of programs without test oracles. In *Proceedings of the 2009 ACM International Symposium on Software Testing and Analysis (ISSTA 2009)*, pages 189–199, 2009.

[11] G. J. Myers. *The Art of Software Testing*. John Wiley and Sons, second edition, 2004. revised and updated by T. Badgett and T. M. Thomas with C. Sandler.

[12] National Institute of Standards and Technology. The economic impacts of inadequate infrastructure for software testing. `http://www.nist.gov/director/prog-ofc/report02-3.pdf`, Gaithersburg, Maryland, USA, 2002.

[13] T. J. Ostrand and M. J. Balcer. The category-partition method for specifying and generating functional tests. *Communications of the ACM*, 31(6):676–686, 1988.

[14] J. Sanders and E. Curran. *Software Quality: A Framework for Success in Software Development and Support*. Workingham: Addison-Wesley, 1994.

[15] L. J. White and E. I. Cohen. A domain strategy for computer program testing. *IEEE Transactions on Software Engineering*, 6(3):247–257, 1980.

[16] H. Zhu, P. A. V. Hall, and J. H. R. May. Software unit test coverage and adequacy. *ACM computing surveys*, 29(4):366–427, 1997.