Postprint of article in *Software Testing, Verification and Reliability* **20** (2): 89–120 (2010)

Finding Failures from Passed Test Cases: Improving the Pattern Classification Approach to the Testing of Mesh Simplification Programs*†

W.K. Chan‡, City University of Hong Kong, Hong Kong

Jeffrey C.F. Ho, wwwins Consulting Hong Kong Limited, Hong Kong

T.H. Tse, The University of Hong Kong, Hong Kong

Abstract

Mesh simplification programs create three-dimensional polygonal models similar to an original polygonal model, and yet use fewer polygons. They produce different graphics even though they are based on the same original polygonal model. This results in a test oracle problem. To address the problem, our previous work has developed a technique that uses a reference model of the program under test to train a classifier. Using such an approach may mistakenly mark a failure-causing test case as passed. It lowers the testing effectiveness of revealing failures. This paper suggests piping the test cases marked as passed by a statistical pattern classification module to an analytical metamorphic testing module. We evaluate our approach empirically using three subject programs with over 2700 program mutants. The result shows that, using a resembling reference model to train a classifier, the integrated approach can significantly improve the failure detection effectiveness of the pattern classification approach. We also explain how metamorphic testing in our design trades specificity for sensitivity.

† This research is supported in part by GRF grants of the Research Grants Council of Hong Kong (project nos. 123207 and 716507), a grant from City University of Hong Kong (project no. CityU 7002324), and a discovery grant of the Australian Research Council (project no. DP0984760). A preliminary version of this paper was presented at the 31st Annual International Computer Software and Applications Conference (COMPSAC 2007) [13].

‡ All correspondence should be addressed to Dr. W.K. Chan at Department of Computer Science, City University of Hong Kong, Tat Chee Avenue, Hong Kong. Tel: (+852) 2788 9684. Fax: (+852) 2788 8614. Email: wkchan@cs.cityu.edu.hk

(a) 100% (b) 70% (c) 30%

Figure 1: Mesh simplification of polygonal models of a Beethoven statue

Keywords: test oracle problem, mesh simplification, non-testable software, metamorphic testing, classification, testing methodology.

1 Introduction

Content-rich software applications with multimedia and graphics subsystems are popular and increasingly complex. These applications accept specifications in formats such as Analyze 7.5 [46] for medical imaging [1], X3D [59] and PLY [56] for image rendering, and MPEG4 [47] for video and audio, and then create the graphics. For real-life interactive graphics-based applications such as scene creation in computer games [17, 36, 63], slow visualization of graphical models is too costly. To address this issue, for instance, computer games often use low-resolution versions of polygonal models to produce shadowing effects or to visualize distant objects [19].

A main category of general techniques to produce these low-resolution versions of polygonal models is *mesh simplification* [16, 19, 23, 38, 64]. Mesh simplification converts a given 3D polygonal model to one with fewer polygons while appearing to be *similar* to the original. For instance, Figure 1 shows three Beethoven statues in different numbers of polygon produced by mesh simplification. The statue in Figure 1(a) has simplified into the ones using only 70% and 30% of the original 3D polygonal model as shown in Figures 1(b) and 1(c), respectively. Nonetheless, different mesh simplification techniques aim to achieve diverse optimization goals such as little memory storage, fastest speed, or best shadow accuracy of objects [37]. They thus produce different changes on the same original polygonal models to create their versions of simplified graphics. It is hard for testers to assess the accuracy of the test output of a program. This results in the test oracle problem. [1]

Two mesh simplification programs may be a variant of each other. They can be of the same kind, or of different kinds, in the taxonomy of simplification algorithms [37]. For instance, a topology-preserving simplification algorithm preserves the outline of the model at every step. If there is a hole in an original model, the algorithm keeps the hole during and after the simplification process. On the other hand, a topology-modifying one may close up holes in the model and merge separate objects into one unit as simplification progresses. We say that a reference model ***resembles***

[1] A *test oracle* is a mechanism to decide whether a program output is correct. When the test oracle is unavailable or too costly to use, it leads to the *test oracle problem* [60].

the implementation under test (IUT) if these two programs belong to the same kind in the taxonomy of simplification algorithms. If the two programs belong to different kinds in the taxonomy, we say that they are ***dissimilar***.

Our previous work [11, 12], known as PAT, has shown empirically that the use of a resembling reference model to train a classifier can be a better pseudo-oracle than the use of a dissimilar reference model. On the other hand, owing to the statistical nature of pattern classification [22], many failure-causing test cases [2] remain undiscovered. This hinders the effectiveness of PAT.

Metamorphic testing (MT) [14] checks whether a set of test cases and their respective program outputs satisfy a data relation. In the simplest form, MT creates a *follow-up* test case based on an *initial* test case. It then compares the test outputs of the initial and follow-up test cases to see whether they may contradict any given necessary relations expected by the correct version of the IUT. Any such contradiction shows a failure. MT calls these necessary relations as *metamorphic relations*. Although MT can be useful, the need of additional test cases becomes a hurdle in employing MT to test content-rich software. Methods to ease this problem are desirable.

We believe that a classifier [22] trained by a reference model provides a complementary helping hand: A test case labeled as failure-causing by a classifier suffices to catch the testers' attention. On the other hand, a passed test case may still be failure-causing. We may treat the test case as an initial test case, and pipe it to the MT module to produce a follow-up test case. We then let the classifier decide the class of the follow-up test case (i.e., the passed class or the failed class). Because a test case marked as failure-causing will receive testers' attention, we need not apply the remaining MT output checking procedure to such a follow-up test case. On the contrary, when a follow-up test case marked as passed, we apply the MT procedure to check whether the passed initial test case, the passed follow-up case, and their test results breach any metamorphic relations. In short, by this scheme, we propose to apply MT to check the outputs of only those test cases that a classifier has marked as passed. This motivates us to propose integrating MT with PAT.

The main contributions of the paper are as follows: It presents a formalized methodology known as PAT 2.0 to integrate analytical and statistical techniques that identify failures for testing mesh simplification programs and their kinds. It reports an experimental study. The empirical result shows that MT significantly complements the classification approach to identifying failures. It also explains why and how MT in our methodology presents a tradeoff between the sensitivity and specificity of a binary classification scheme.

We organize the rest of the paper as follows: Section 2 reviews related work on the testing of software with graphical interfaces. Section 3 gives the technical background of the work. Section 4 presents our methodology. Section 5 presents our experimental setup, results, and threats to validity. We then discuss our findings and explain how MT trades specificity for sensitivity in Section 6. Finally, we conclude the paper in Section 7.

[2] A failure-causing test case is a test case that reveals a failure from an IUT.

2 Related Work

We review related work that uses machine learning approaches as pseudo-oracles, as well as related work on metamorphic testing and other approaches to ease the test oracle problem. For brevity, we shall focus on the testing of software with graphical interfaces.

Berstel and colleagues [4] design the VEG language to describe graphical user interfaces and show that model checkers may verify properties against a specification written in VEG without referring to the source program. Our approach does not rely on the source code of the program under test either, but our work involves dynamic analysis whereas their technique is static. D'Ausbourg and colleagues [18] support the formal design of operations in user interface systems by a software environment. One may use the technique presented in [4] to verify such a design. Memon and colleagues [44] use a test specification of internal object interactions as a means of detecting inconsistencies between the test specification and the resulting execution sequence of events for each test case. Such an approach is also popular in conformance testing of telecommunication protocols. Sun and Jones [57] propose a similar approach for test harnesses. Memon and colleagues [43] further evaluate several types of pseudo-oracle for GUIs. Their results suggest the use of simple pseudo-oracles for large test sets and complex pseudo-oracles for small test sets. Our work does not explore such kinds of tradeoff, but integrates a complex pseudo-oracle (classification) with a simple one (metamorphic relation).

There are other approaches to testing programs with graphical outputs. gDEBugger[3] checks whether a list of commands issued by an application conforms to the usual underlying graphics visualization application programming sequences used in OpenGL [54]. As explained in [5, 11, 12], however, many different sequences of commands may render the same graphical image. Checking whether a particular sequence of commands has been used to produce a particular image may not be fully reliable. To test programs with interfaces with virtual reality applications, Bierbaum [5] proposes a framework to record selected intermediate states of program executions and contrast them against the expected ones. Following PAT [11, 12], we do not use the internal states of the program under test. Cheung and colleagues [15] and Mayer [40] both use explicit statistical formulas such as mean and distributions to check whether a test output carries desirable characteristics. Mayer and Guderlei [41] evaluate the impact of different metamorphic relations on Java programs that compute numerical determinants. Their study agrees with our previous work [30] that metamorphic testing is useful in easing the test oracle problem.

Researchers also have studied the test oracle problem in other contexts. Ostrand and colleagues [49] propose an integrated environment to ease testers to review and modify their test scripts. Dillon and Ramakrishna [21] prune the search space of test oracles constructed from a specification. Baresi and colleagues [3] add assertions [45] to programs to check their intermediate states. Peters and Parnas [50] propose to construct precise program documentations and generate oracles from them.

Apart from statistics approaches, there are analytical approaches such as golden version [6] and assertion checking [45]. Nonetheless, as we have explained when describing the test oracle problem, different mesh simplification algorithms produce similar but diverse outputs. Using a

[3] Available at http://www.gremedy.com/.

golden version may not help in this case. Assertion checking verifies whether a program state or the output of a program execution satisfies an expected condition. Many industrial applications, such as the popular Microsoft .NET framework, have successfully applied assertion checking.

Applying pattern classifications to ease the test oracle problem is not new. Last and colleagues [34, 58] train a classifier to augment the incomplete specification of a legacy system, and treat the legacy system as a golden version. As we have explained, golden versions are often unavailable for mesh simplification programs. Podgurski and his research group classify failure cases into categories by machine learning [51] and then refine the categories using the classification tree technique [24]. Bowring and colleagues [7] apply machine learning to regression testing of a consecutive sequence of minor revisions of the same program to identify failures in subsequent versions. Their approach is similar to the reference model approach proposed by us [11, 12]. However, their approach requires the source code of the program under test. Our previous work does not have this requirement, but needs to produce mutants of the reference models, which are reusable when testing other mesh simplification programs of the same kind. Another pattern classification approach in [10] does not use reference models.

3 Background

Our methodology builds on top of PAT [12] and extends it with an analytical module, for which we choose metamorphic testing. In this section, we present the background of these two techniques. To ease our presentation, we rename our previous technique [12] from PAT to PAT 1. Readers who are familiar with these background techniques may skip this section and go directly to Section 4.

3.1 PAT 1

In our previous work [11, 12], we have developed a methodology now known as PAT 1, which stands for "Pattern classification to Automatic reference oracles for the Testing of mesh simplification programs". It trains a classifier using an IUT's reference model, and then uses the classifier [22] to identify failures from the test outputs of the IUT. In this section, we review PAT 1.

As we have described in Section 1, it is hard to identify a failure from the test results of a mesh simplification program. On the other hand, to train a classifier (denoted by C) for binary classification [61] of test results, we need a method to produce training samples for the passed and failed classes.

To produce training samples for the passed class, we run an IUT's reference model over a set of 3D polygonal models to produce image outputs. We then extract values from such an output for a vector of image features, and use all such value vectors as training samples. In this way, we obtain a dataset of training samples for the passed class. To ease our subsequent discussion, we denote the dataset by S_P.

To produce training samples of the failed classes, we create program mutants [2, 48] from the reference model. We first run the program mutants over the same set of 3D polygonal models to produce image outputs. We then extract values from each image output for the same vector of

image features. We include the value vector into in the dataset (denoted by S_F) of training samples for the failed class only if S_P does not already have this value vector.

In the testing phase, we run the IUT over a test case. Like the training phase, we extract values from the corresponding image output for the same vector of image features. The classifier C then decides the class of the value vector (namely, failed or passed). Finally, we mark the test case as failure-causing if the value vector belongs to the failed class. Otherwise, we mark the test case as passed.

Chan and colleagues [12] has formalized the PAT 1 procedure, which is also listed as follows:

> Let C be a classifier to test an implementation under test *IUT* with a reference model R. Let $\mathcal{M} = \{m_1, m_2, \ldots, m_i, \ldots, m_k\}$ be a set of 3D polygonal models, serving as test cases. Executing R over $\mathcal{M}$ will produce a set of outputs $\{R(m_1), R(m_2), \ldots, R(m_k)\}$. Suppose the program mutants [20, 48] of R are denoted by $\{\overline{R_1}, \overline{R_2}, \ldots, \overline{R_u}\}$. Executing each $\overline{R_j}$ of these mutants over $\mathcal{M}$ will produce a corresponding set of outputs $\{\overline{R_j}(m_1), \overline{R_j}(m_2), \ldots, \overline{R_j}(m_k)\}$.
>
> Let $\langle f_1, f_2, \ldots, f_v \rangle$ be a list of classification feature extraction functions that extracts features from input polygonal models and program outputs. Given an input model m_i, the reference program R, and the output $R(m_i)$, the above list of functions will extract a list of features $\langle f_1(m_i, R, R(m_i)), f_2(m_i, R, R(m_i)), \ldots, f_v(m_i, R, R(m_i))\rangle$, known as a *vector of extracted features*.
>
> Similarly, for each mutant $\overline{R_j}$, the list of functions will produce a corresponding vector of extracted features $\langle f_1(m_i, \overline{R_j}, \overline{R_j}(m_i)), f_2(m_i, \overline{R_j}, \overline{R_j}(m_i)), \ldots, f_v(m_i, \overline{R_j}, \overline{R_j}(m_i))\rangle$. If the vector of extracted features produced from mutant $\overline{R_j}$ is identical with that produced from R, PAT 1 will discard the vector. We refer to the remaining vectors as *non-equivalent mutation vectors*.
>
> PAT 1 labels every such vector of extracted features as *passed* and every such non-equivalent mutation vector as *failed*. PAT 1 uses all these labeled vectors to train the classifier C for binary classification.
>
> To test *IUT*, PAT 1 executes it over a set of test cases, and constructs the vectors of extracted features for *IUT* using the above scheme but replacing R by *IUT*. PAT 1 then passes each of such vectors of *IUT* to the trained classifier C, and let the classifier label the vector. A vector labeled as *passed* means observing no failure, and thus, the corresponding test cases to produce the vector will be marked as *passed*. On the other hand, a vector labeled as *failed* indicates a failure, and the corresponding test cases to produce the vector will be marked as *failure-causing*.

Two specific designs in PAT 1 are the use of reference model and the use of black-box features. Based on empirical evaluation, we have also made a recommendation in [12] to testers when using PAT 1. We discuss them further below.

Reference model. PAT 1 requires a reference model R. As we have described in Section 1, a reference model may resemble the IUT or be dissimilar to it. To know whether a reference model resembles or is dissimilar to the IUT in the taxonomy of simplification algorithms [37], we seek external advices. We have consulted the members of the graphic research groups at The University of Hong Kong and searched the literature (such as [53]). To our best knowledge, expert judgment is still necessary to decide the classification. PAT 1 presumes to know whether a reference model resembles the IUT.

Black-box features. A classification feature extracted from the program runtime behavior may be black-box (for example, the mean brightness of an image), white-box (for example, the number of branch statements covered by a test case), or a combination of them. Coincidental correctness [28] occurs when a program execution has activated a fault to become an error, yet the error does not propagate to any output to become a failure. However, before finding out the faults on an execution path, knowing whether coincidental correctness has occurred in the execution is difficult.

Coincidental correctness thus distinguishes using black-box information from using white-box information to stand for a classification feature for testing purposes. A white-box feature generally needs the knowledge of program states. Owing to the occurrence of potential coincidental correctness, the involved program states of a passed test case can be abnormal. To use such a feature to train a classifier for the passed class, testers should confirm the used white-box information indeed as expected. Otherwise, they may confuse the classifier to serve as a pseudo-oracle.

On the other hand, coincidental correctness does not affect the output of a program execution. Thus, if we extract features from an image output, coincidental correctness will not affect these features. Therefore, PAT 1 uses black-box features.

Recommendation. In PAT 1, the reference model R may resemble the IUT or be dissimilar to it. In [11, 12], we have empirically evaluated that using a resembling reference model as the reference model R can be significantly more effective to identify program failures than using a dissimilar one. Thus, in [11, 12], we recommend testers to use a resembling reference model whenever it is available as a means to train a classifier under PAT 1.

Empirical Results. We also revisit the results of PAT 1 in [12]. The purpose of this review is to let readers know the effectiveness of PAT 1 to identify failures for testing mesh simplification programs. We first recall that in PAT 1, a classifier only has two classes of outcomes, namely labeling a test case as passed or failure-causing. In other words, PAT 1 is a binary classification scheme.

According to [12], the average effectiveness of PAT 1 when using a resembling reference model (denoted by PAT 1.r) is 69.0%. However, when using a dissimilar reference model, the average effectiveness of PAT 1 (denoted by PAT 1.d) becomes 33.4%. In other words, PAT 1.r can be 106.5% more effective than PAT 1.d.

We have reviewed PAT 1. In the next section, we will review metamorphic testing, which is another module in our new methodology.

3.2 Metamorphic Testing

This section revisits metamorphic testing (MT) [14]. A central idea of MT is to check the expected necessary properties of the program under test that relate multiple test cases and their test results with a view to revealing failures. It captures such a necessary property as a metamorphic relation.

A *metamorphic relation* (*MR*) [9, 14] is a relation over a set of distinct inputs and their corresponding outputs of the target function p that the program P under test aims to implement. Let us take the sine function for example: For any inputs x_1 and x_2 such that $x_1 + x_2 = \pi$, we must have $\sin x_1 = \sin x_2$.

Given a test case $x_1 = \pi/6$, a tester will obtain a follow-up test case x_2 ($= 5\pi/6$) based on the relation $x_1 + x_2 = \pi$. For instance, testers may use a constraint solving approach or implement a program to generate x_2 by subtracting x_1 from the constant π.[4]

By executing the sine program over both x_1 and x_2, the tester will obtain the corresponding test results, say, 0.5000 and 0.5004, respectively. Then, the tester checks whether the two outputs satisfy the relation $\sin x_1 = \sin x_2$, which means whether 0.5000 is equal to 0.5004. If the equality does not hold, which is the case for this particular example, MT detects a failure.

A metamorphic relation is as follows [9]:

$$\begin{aligned} &MR\text{: If } r(x_1, x_2, \ldots, x_k, p(x_1), p(x_2), \ldots, p(x_k), x_{k+1}, x_{k+2}, \ldots, x_n), \\ &\quad \text{then } r'(x_1, x_2, \ldots, x_k, x_{k+1}, \ldots, x_n, p(x_1), p(x_2), \ldots, p(x_k), p(x_{k+1}), \ldots, p(x_n)) \end{aligned}$$

Here, $x_1, x_2, \ldots, x_k$ are initial test cases; $x_{k+1}, x_{k+2}, \ldots, x_n$ are follow-up test cases; $p(x_i)$ is the expected output of the function p over x_i; and r and r' are relations.

Testers should study the problem domain to define metamorphic relations. This is akin to requirements engineering, in which requirements engineers rather than automatic engines are essential to elicit and specify the system requirements.

The following paragraph shows the definition of metamorphic testing.

Definition 1 (Metamorphic Testing) [9] *Let P be an implementation of a target function p. The* metamorphic testing *of the metamorphic relation*

$$\begin{aligned} &MR\text{: } \textit{If } r(x_1, x_2, \ldots, x_k, p(x_1), p(x_2), \ldots, p(x_k), x_{k+1}, x_{k+2}, \ldots, x_n), \\ &\quad \textit{then } r'(x_1, x_2, \ldots, x_k, x_{k+1}, \ldots, x_n, p(x_1), p(x_2), \ldots, p(x_k), p(x_{k+1}), \ldots, p(x_n)) \end{aligned}$$

involves the following steps: (1) *Given a series of initial test cases* $\langle x_1, x_2, \ldots, x_k \rangle$ *and their respective results* $\langle P(x_1), P(x_2), \ldots, P(x_k) \rangle$*, generate a series of follow-up test cases*

[4] For instance, in [30], we have experimented to request developers to specify metamorphic relations and implement programs to generate follow-up test cases. Interested readers may contact us to obtain the source codes of these implementations of metamorphic relations.

$\langle x_{k+1}, x_{k+2}, \ldots, x_n \rangle$ *according to the relation* $r(x_1, x_2, \ldots, x_k, P(x_1), P(x_2), \ldots, P(x_k), x_{k+1}, x_{k+2}, \ldots, x_n)$ *over the implementation P.* (2) *Check the relation* $r'(x_1, x_2, \ldots, x_k, x_{k+1}, \ldots x_n, P(x_1), P(x_2), \ldots, P(x_k), P(x_{k+1}), \ldots, P(x_n))$. *If* r' *is evaluated to be false, then the metamorphic testing of MR reveals a failure.*

In the next section, we will describe our methodology and define the research question.

4 Our Methodology: PAT 2

This section proposes a testing methodology that combines PAT 1 [11, 12] with metamorphic testing [14]. It studies the integration of statistical techniques and analytical techniques for alleviating the test oracle problem. We call the methodology PAT 2. As the name implies, it extends PAT 1.

In PAT 1, a trained classifier will label a test case as *failure-causing* or *passed*. A test case marked as failure-causing would catch the attention of testers. For instance, Scenario (A) in Figure 2 sketches an example of the use of PAT 1 to identify a failure-causing test case. On the top left corner of Figure 2, there is a sample test case labeled as *m1*. As indicated by its comment line, *m1* specifies a 3D polygonal model of a wind direction sign. The visual output of the IUT over *m1* (labeled as *Output of IUT(m1)*) is shown next to the test case for readers' reference. Adjacent to the visual output, we also show a zoom-in image of the chest part of the cock symbol, and highlight the failure. As we have described in Section 1, this type of failure may intermix with inaccurate expectations on the output by a tester. Thus, the failure can be hard to be observed manually. This particular scenario illustrates that PAT 1 can help identify a failure successfully, so that we need not apply PAT 2.

On the other hand, because of the statistical nature of a classifier, test outputs marked as *passed* by PAT 1 may still be failures. Thus, after classifier has checked that an (initial) test case does not reveal any failure, PAT 2 pipes the test case (and its test output) to an MT module to conduct further checking. Specifically, the MT module will construct a follow-up test case based on the initial test case. Scenario (B) in Figure 2, for instance, shows that there is a test case *m1* which has passed the PAT 1 phase. PAT 2 proceeds to construct a follow-up test case *m2*. For this particular scenario, the test case *m2* is constructed using the metamorphic relation MR_3 (see Section 5.1.4), which expects to turn the imagine output upside down. For example, we may observe from the test case *m2* in Figure 2 that the *y*-component of each vertex has changed in sign, compared with the corresponding vertex of *m1*.

We run the IUT over the follow-up test case to obtain the test result and use the classifier above to label the test case. If PAT 1 shows a failure in the follow-up test case, the latter should receive testers' attention. By a token similar to the handling of *m1*, we do not apply MT's output checking procedure on *m2*, as illustrated in Scenario (B) of Figure 2.

If the follow-up test case is labeled as passed, PAT 2 further compares the initial and follow-up test cases and their test results to check whether they breach the given metamorphic relations. This is shown in Scenario (C) of Figure 2. In the scenario, we have a decision box labeled as "m2 = m1 (but inverted)?" It is the MT output checking procedure according to MR_3 (see Section 5.1.4),

which compares whether the features vector of the output image from the IUT over *m2* agrees with the features vector obtained by inverting the image output from the IUT over *m1*. On the top right corner of Figure 2, we also show the output of the IUT over *m2* (labeled as *Output of IUT(m2)*) and the enlarged part of the cock chest. We may observe between these enlarged chests in the outputs of *m1* and *m2* that they are not identical, which breach MR_3. Therefore, PAT 2 labels the pair of test cases (*m1* and *m2*) as failure-causing.

The PAT 2 methodology is as follows.

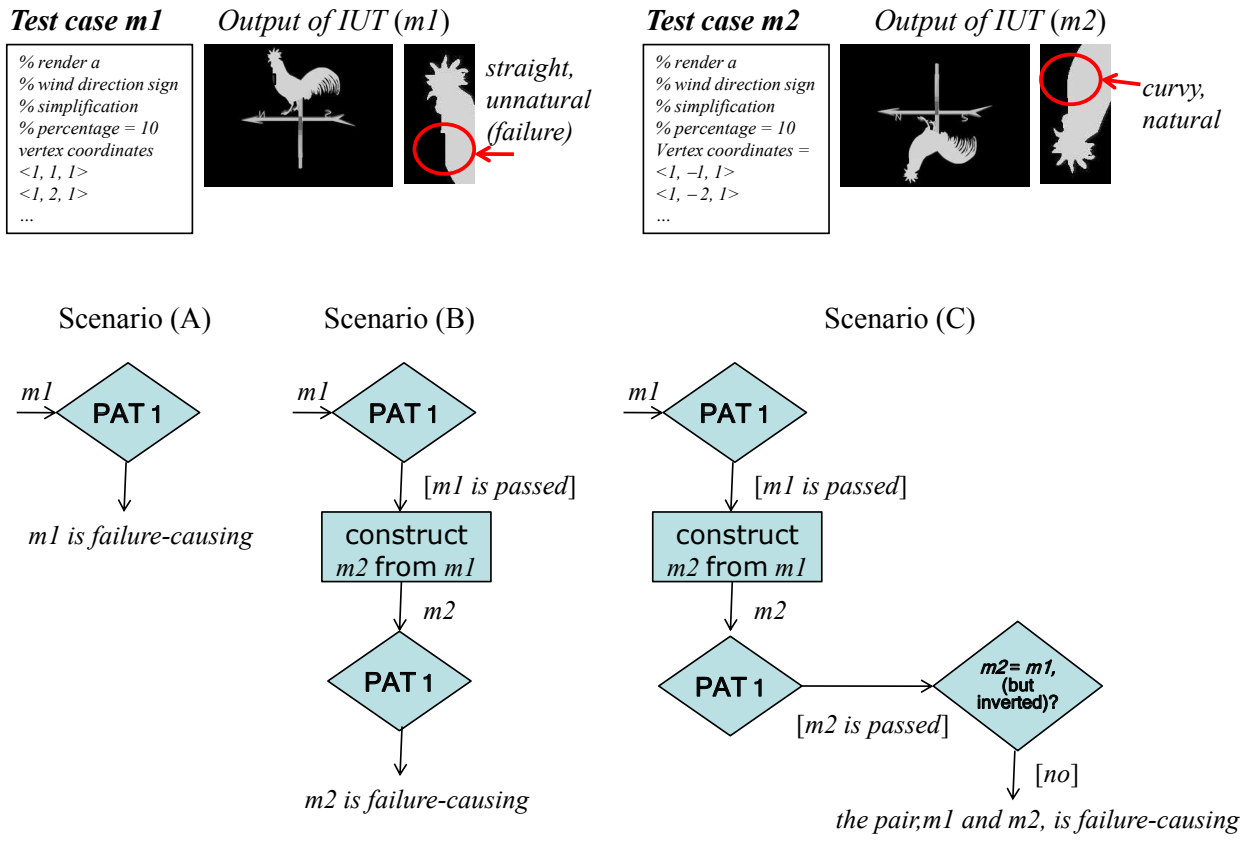


Figure 2: Blueprint of the Methodology PAT 2.

(1) Given a test case m_i for testing a mesh simplification program P. PAT 1 labels m_i as either *passed* or *failure-causing*.

(2) If PAT 1 labels m_i as *failure-causing*, exit from the methodology.

(3) If PAT 1 labels m_i as *passed*, we further apply the MT module for further verification. Let $\{MR_1, MR_2, \ldots, MR_n\}$ be a set of metamorphic relations of the expected function of the program P. To simplify our presentation but without loss of generality, let us assume that the implementation of each MR_k accepts one initial test case and produces one follow-up test case.

	Using Resembling Reference Model	Using Dissimilar Reference Model	Arbitrary Reference Model
PAT 1 without piping to MT	PAT 1.r	PAT 1.d	PAT 1.0
PAT 1 with piping to MT	PAT 2.r	PAT 2.d	PAT 2.0

Table 1: Different Naming Convention for PAT 1 and PAT 2

Applying the given implementations of the metamorphic relations to a test case m_i will produce a set of follow-up test cases $\{MR_1(m_i), MR_2(m_i), \ldots, MR_n(m_i)\}$. Executing the IUT over these follow-up test cases will produce the outputs $\{P(MR_1(m_i)), P(MR_2(m_i)), \ldots, P(MR_n(m_i))\}$.

(4) We use the classifier of PAT 1 again to determine whether the follow-up test cases reveal a failure. For follow-up test cases labeled as failure-causing, exit from the methodology.

(5) We use the implementation of each applicable metamorphic relation $\{MR_1, MR_2, \ldots, MR_n\}$ to compare the initial test case m_i, the follow-up test cases $MR_1(m_i), MR_2(m_i), \ldots, MR_n(m_i)$, and their test outputs. If MR_k is breached for some $k \in \{1, 2, \ldots, n\}$, we label the test case m_i and the follow-up test case $MR_k(m_i)$ as failure-causing.

In this way, PAT 2 saves the effort in applying MT by only checking the test cases that PAT 1 classify as *failure-causing*. Still, it is uncertain whether the extra step of applying MT is worth the effort. A research question thus arises:

> **RQ1:** *During the testing of mesh-simplification software, how much improvement in the effectiveness of failure identification will result by piping the results of a pattern classification approach to MT?*

As we have explained in Section 3.1, a reference model R in PAT 1 can be a resembling reference model or a model of a dissimilar kind. To evaluate PAT 2 rigorously, we thus want to study whether using a resembling reference models or a dissimilar reference model as the reference model R in PAT 1 to have any significant impacts on the PAT 2 methodology.

In the sequel, to ease our presentation, we make the following naming convention to differentiate various combinations of reference model, PAT 1 and MT. We denote the version of PAT 1 that uses a resembling reference model by PAT 1.r. By the same token, we denote the version of PAT 1 that uses a dissimilar reference model by PAT 1.d. Because PAT 2 builds atop of PAT 1, we denote the versions of PAT 2 based on PAT 1.r and PAT 1.d by PAT 2.r and PAT 2.d, respectively.

In some practical situations, testers may not decide whether a reference model resembles the IUT, but just select an arbitrary reference model. Hence, we also study whether it is useful to pipe the results of PAT 1 to MT if the pair of reference model and IUT is unclear to be resembling or dissimilar. (That is, not knowing whether PAT 1.r and PAT 1.d is being applied). We denote this version of PAT 1 by PAT 1.0. Following the naming convention above, we denote the version of PAT 2 building on top of PAT 1.0 by PAT 2.0. We show the naming convention in Table 1.

Thus, we define the following null hypotheses for our further study:

H_1: There is no significant difference between PAT 1.0 and PAT 2.0 in failure identification ability.

H_2: There is no significant difference between PAT 1.r and PAT 2.r in failure identification ability.

H_3: There is no significant difference between PAT 1.d and PAT 2.d in failure identification ability.

H_4: There is no significant difference in the improvement between PAT 2.r over PAT 1.r and PAT 2.d over PAT 1.d in failure identification ability.

Following the advice given in [12], we do not hide our interests to study whether PAT 2 can be more effective than PAT 1, which is a step toward understanding why the technique may or may not be useful. Let us explain it further.

Rejecting the hypothesis H_1 helps demonstrate that connecting a statistical technique to an analytical technique can be useful to identify failures. On the other hand, if we could accept the hypothesis H_1 and yet reject either the hypothesis H_2 or the hypothesis H_3, the results may indicate the integration being multi-modal. This may help define further research questions on understanding the reasons on why such integration shows a multi-modal behavior in the future. Another case is that we could accept all of H_1, H_2, and H_3. Although such a case shows a negative result, yet it provides evidences that connecting statistical and analytical techniques to ease the test oracle problems is more challenging than what we know at this stage. Lastly, if we could reject both H_2 and H_3, we then want to know whether the improvement delivered by PAT 2.r over PAT 1.r can be more significant than that delivered by PAT 2.d over PAT 1.d, which is H_4. This helps give practical advices to testers.

In the next section, we will describe the empirical study and analyze the findings.

5 Empirical Study

In this section, we present an empirical evaluation of the research question. Section 5.1 will present the setup of the experiment. In Section 5.1.1, we will describe the subject programs used in the experiment. Section 5.1.2 presents how the test cases and datasets for the previous classification experiment in [11, 12] are created. Section 5.1.3 introduces and explains the metrics to evaluate the results of the current study. In Sections 5.1.4 and 5.2, we present the metamorphic relations of the subject programs and the procedure of the experiment. Finally, we analyze the results of the current study in Section 5.3.

5.1 Experimental Setup

In this section, we describe the setup of our empirical study. The study is built on top of the previous experiments conducted in our previous work [11, 12]. We use their classification dataset as the starting point of the current experiment. In the present study, we construct a few metamorphic relations and pipe the test cases marked as *passed* in the dataset to the MT module to identify additional failures.

5.1.1 Subjects of the Experiment

We use the Java programs studied in [11, 12] as this would enable us to compare the findings with PAT 1. Each program implements a distinct mesh simplification algorithm: Melax's simplification algorithm [42], the quadric algorithm [25], and a quadric algorithm weighted by the areas of surrounding triangles [25]. We denote their implementations in the experiment by *Melax*, *Quadric*, and *QuadricTri*, respectively. Each of these algorithms accepts a 3D polygonal model and outputs a simplified one.

(1) *Melax* measures the cost of each edge in a polygonal model as a product of its length and curvature. It iteratively picks the edges with the lowest costs to remove until the model has reduced to the required number of polygons.

(2) *Quadric* contracts pairs of vertices rather than edges, so that unconnected regions in a polygon model may merge. It approximates contraction errors by quadric matrices. It picks the pairs of vertices with the lowest costs to remove until the model has reduced to the required number of polygons.

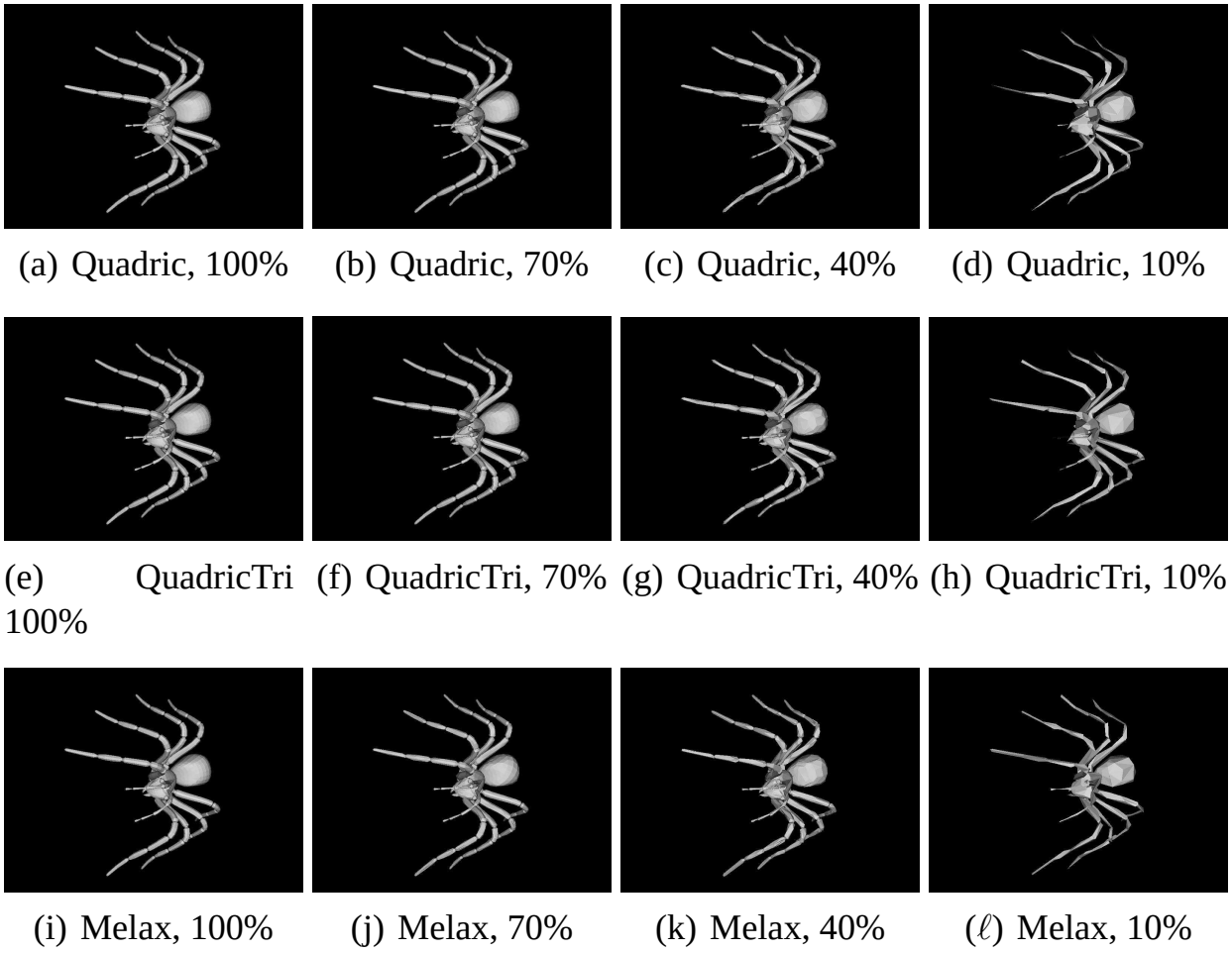
(3) *QuadricTri* improves on *Quadric* by considering also the sizes of triangles around vertices during contraction.

Quadric and *QuadricTri* are topology-modifying mesh simplifications [37]. They resemble each another, and are dissimilar from *Melax*, which is of the topology-preserving kind [37]. Figure 3 shows a spider simplified by these programs for readers' reference. Take Figure 3(d) for example. The image shows a spider image output produced by *Quadric* using 10% of polygons of the given 3D polygon model of the spider. Other sub-figures in Figure 3 can be interpreted similarly. We may observe from Figure 3 that the outputs of the subject programs at any given simplification percentage are quite close to one another.

Each of the three subject programs serves two roles in the experiment of [11, 12]. Take *Quadric* for illustration. The experiment has used *Quadric* as a reference model to train up the classic C4.5 classifier. The trained classifier will mark the test outputs of the other two subject programs (*QuadricTri* and *Melax*) serving the role of the IUT. (Section 5.1.2 will describe the test cases used in experiment.) In turn, when we use *Melax* or *QuadricTri* to train the classifier, *Quadric* will act as the IUT.

Each program accepts a 3D polygonal model file in the standard PLY format [56] with an integer (from 0 to 100) indicating the target percentage of polygons that will remain after mesh simplification. We call this integer parameter as *simplification percentage*. For instance, if the value of the simplification percentage is zero, it shows only the background. Similarly, when the value of the simplification percentage is 100, it will show the original model without any simplification effect. The backgrounds of all outputs are black in color. Each program fits the 3D polygonal model in an area between $(-1, -1, -1)$ and $(1, 1, 1)$, centered at $(0, 0, 0)$. The image resolution is standardized to 800 pixels $\times$ 600 pixels.

To ease our presentation, we treat the simplification percentage as an attribute of the input 3D polygonal model. Therefore, in this paper, we simply refer a 3D polygonal model as an input to a subject program.



(a) Quadric, 100% (b) Quadric, 70% (c) Quadric, 40% (d) Quadric, 10%

(e) QuadricTri 100% (f) QuadricTri, 70% (g) QuadricTri, 40% (h) QuadricTri, 10%

(i) Melax, 100% (j) Melax, 70% (k) Melax, 40% (ℓ) Melax, 10%

Figure 3: Sample Results of Different Mesh Simplification Programs.

5.1.2 Test Cases and Dataset of PAT 1

Chan et al. [11, 12] use a supervised machine learning approach to label test cases in two categories: *passed* and *failure-causing*. We use their results in our empirical study. We describe in brief their experiment to produce the test cases and datasets in this section.

The previous experiment in [11, 12] executes a set of 44 open-source 3D polygonal models [5] with up to 17,000 polygons in each reference model. We denote the set of models by Θ. We observe that viewing a spider from the front is definitely different from viewing it from the rear. Thus, even for the same input model, a failure from one perspective may not occur in another perspective image. In order to better utilize the 3D polygonal models, the previous experiment rotates each one in 22 different orientations. They correspond to rotating a model about the *x*-axis every 22.5 degrees and rotating about the *y*-axis every 45 degrees. Each orientation is further

[5] Downloaded from http://www.melax.com/polychop/lod_demo.zip. According to this source, they are "big demo[s] . . . to convince skeptical visitors".

augmented with 11 different simplification percentages (from 0 to 100 with increments of 10) to produce 11 3D polygonal models. In other words, we have created a pool of $44 \times 22 \times 11 = 10,648$ test cases. To ease our discussion about the experiment setup, we denote this pool of test cases by *TP*. We further use $TP(m)$ to denote the entire subset of test cases of *TP* produced by the same 3D polygonal model *m* (which is in Θ).

To collect training samples S_P for the *passed* class, it executes every subject program over every test case in *TP* to produce an output. It then extracts black-box features from every such graphics output. These black-box features are as follows. (1) Change of ratios of major and minor image frequencies under fast Fourier transform [27]. (2) The average brightness of the graphic. All these vectors of black-box features are put in S_P.

Informally, a simplified polygonal model will have a higher overall frequency value than the original polygonal model. This is because when fewer polygons are used to model an image, smaller amounts of image frequencies of the original model will remain in the simplified version. The stronger the strength, the more it will contribute to the image. Signals with major contributions are low frequency signals contributing major image frequencies of the original model. Signals with minor contributions are high frequency signals contributing minor image frequencies of the original model. Hence, we sort the image frequencies according to signal strengths and compute the mean and the mean plus or minus one (two and three) standard deviation(s). By covering up to three standard deviations, the effect of over 99% of the frequencies in an image are considered in the previous experiment. The average brightness feature is to remedy the use of ratio as the first classification feature, which eliminates the effect of any changes that happen to be proportional.

To collect training samples S_F for the *failed* class, it uses MuJava [39] to generate program mutants from every subject program. A total of 3,060 non-equivalent mutants are created, and the previous experiment uses all of them to produce experimental results. It executes every of these mutants over every test case in *TP* to create a vector of non-equivalent mutations. The experiment discards the vector if the vector already exists in S_P. In total, the experiment executes the mutants with more than 440,000 program executions. All such vectors of non-equivalent mutations form the dataset S_F. Table 2 shows the numbers of mutants of the three subject programs used in [11, 12].

Melax	*Quadric*	*QuadricTri*
401	1,122	1,187

Table 2: Numbers of Mutants Used.

The experiment randomly picks one polygonal model *m* from Θ and picks one subject program *R* to train the classical C4.5 classifier [22]. It selects from S_P all the training samples produced from executing *R* over $TP(m)$ as the passed training samples of *R*. Similarly, it selects from S_F all the training samples produced by executing the program mutants of *R* over $TP(m)$ as the failed training samples of *R*. It applies the two selected sets of training samples to train the C4.5 classifier.

In the testing phase, the previous experiment does not use any samples of $TP(m)$ from S_P. It executes the IUT (which is another subject program different from the above reference model) over all the remaining samples in S_P. For instance, if we use *Quadric* to train the classifier, then either *QuadricTri* or *Melax* (but not *Quadric*) can be the IUT. In other words, we let the classifier label

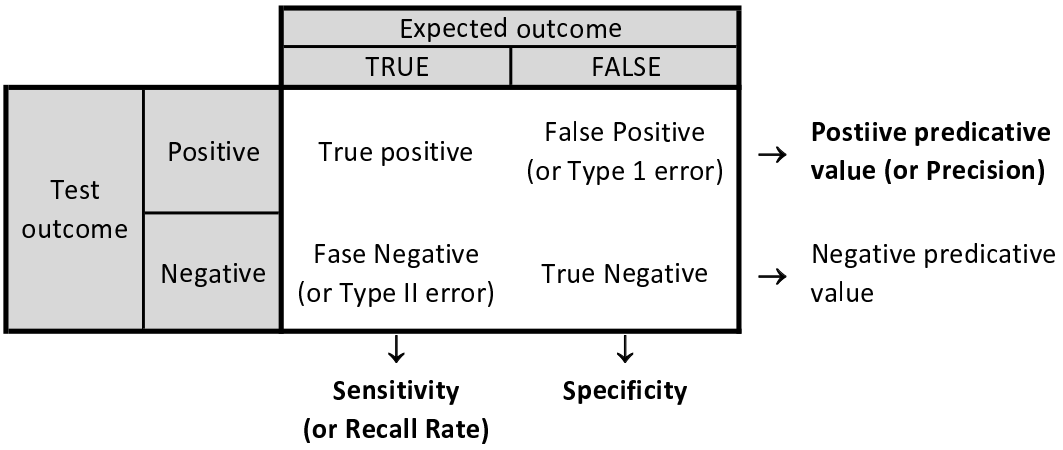


Table 3: Relationships among Sensitivity, Specificity, Recall Rate and Precision

the test verdicts of individual test cases produced from unseen input models for another subject program. We repeat the same experiment using the test cases of two input models to training the classifier, followed by three, four, and finally five input polygonal models. We repeat the entire procedure for every pair of subject programs.

5.1.3 Effectiveness and ER-Score

To evaluate a technique, we need some metrics. In statistics, machine learning, and medical research, **sensitivity** and **specificity** are a standard pair of measures for binary classification [29, 33]. Sensitivity measures how well a binary classification correctly tests a condition whereas specificity measures how well the binary classification correctly identifies the negative cases [29, 33]. In some disciplines such as information retrieval, sensitivity is known as the **recall rate**. However, specificity is not the same as **precision** [33]. To help readers appreciate the differences between sensitivity/specificity and recall/precision, we use a confusion matrix [33] in Table 3 (adapted from [62]) for illustration. We can observe from the table that both sensitivity and specificity evaluate the experiment results along the dimension of expected outcomes (as indicated by the direction of arrows). On the other hand, recall rate and precision evaluate the experiment results along different dimensions, with an overlapping cell (true positive). To cater for the software engineering community and other general audiences, we use sensitivity and specificity as the terminology in this paper.

Sensitivity and specificity are defined in statistics as follows:

$$Sensitivity = \frac{no.\ of\ true\ positives}{no.\ of\ true\ positives\ +\ no.\ of\ false\ negatives} \times 100\%$$

$$Specificity = \frac{no.\ of\ true\ negatives}{no.\ of\ true\ negatives\ +\ no.\ of\ false\ positives} \times 100\%$$

To cast the two measures into software testing, we should map *true positives*, *true negatives*, *false positives*, and *false negatives* to the binary classification used in our experiment, and the

mapping is as follows. Let $\mathcal{E}$ be the expected classifier, and $\mathcal{A}$ be the actual classifier used. The four cases are as follows.

(1) A true positive is a test case such that both $\mathcal{E}$ and $\mathcal{A}$ label the test case as failed. In other words, a failure-causing test case is correctly identified.

(2) A true negative is a test case such that both $\mathcal{E}$ and $\mathcal{A}$ label the test case as passed. In other words, a passed test case is correctly identified.

(3) A false positive is a test case such that $\mathcal{E}$ labels the test case as passed, but $\mathcal{A}$ labels the test case as failed. In other words, a passed test case has been mistakenly classified as failure-causing.

(4) A false negative is a test case such that $\mathcal{E}$ labels the test case as failed, but $\mathcal{A}$ labels the test case as passed. In other words, a failure-causing test case has been missed by a testing technique.

Because we are testers, we interest in studying the effectiveness and false alarms of a testing technique. We thus use more tester-friendly terminologies to redefine sensitivity and specificity. We further define an ER-score as a measure to combine sensitivity and specificity.

$$\begin{aligned} \textit{Effectiveness} &= \textit{Sensitivity} \\ \textit{Robustness} &= \textit{Specificity} \\ \textit{ER-score} &= \textit{Effectiveness} \times \textit{Robustness} \end{aligned}$$

Intuitively, **effectiveness** is the percentage of failure-causing test cases that have been correctly classified by a binary classification scheme. Similarly, **robustness** is the percentage of passed test cases that have been correctly classified by the same scheme. We deem that testers generally would like to maximize both the failed test cases and passed test cases to be correctly classified. As a result, a *higher value* in either measure means a *better result* in a testing experiment.

The **ER-score** is a product of effectiveness and robustness. We use this metric to combine effectiveness and robustness into one value. This score carries the properties that we consider important in evaluating test experiments. When the effectiveness is zero, the ER-score must also be zero, irrespective of the robustness. Similarly, when the robustness is zero, the ER-score must also be zero. On the other hand, only when both effectiveness and robustness are 1, the ER-score can attain its maximal value. Lastly, when effectiveness (or robustness) remains unchanged, the ER-score will vary proportionally to robustness (or effectiveness). Thus, a higher ER-score also indicates a better result.

5.1.4 Metamorphic Relations

Researchers in metamorphic testing advocate the use of simple metamorphic relations to ease the test oracle problem. To follow this advice, we use three simple metamorphic relations in the experiment to check the *passed* test cases produced in the PAT 1 phase. As defining an adequate set of metamorphic relations is still an open problem, we pick the following generic metamorphic relations so that they are not tied to any particular simplification strategy.

- The first metamorphic relation (MR_1) checks the size of the bounding box rendered from an initial test case against that from the non-simplified 3D polygonal model (that is, when the simplification percentage is 100). This is akin to a common practice in assertion checking to check the size of the bounding box after each iteration. As the metamorphic relation is graphics-based, we depict the idea of MR_1 in Figure 4. [6]

  To present the metamorphic relation, we need some helper functions. Let u be a function accepting an image and returning an outline of a shape in the image. Further, let *noScale* be a function accepting a 3D polygonal model and returning the 3D polygonal model with simplification percentage being 100. Given an image $P(m)$ produced by a program P over an input m. The metamorphic relation is as follows:

  $$u(P(m)) \subseteq_c u(P(noScale(m))),$$

  where $\subseteq_c$ is a two-polygon containment relation [26], which asserts that $u(P(m))$ should be within $u(P(noScale(m)))$.

  In the experiment, we use the built-in function of Adobe Photoshop 7.0 batch processing to produce the outline of the shape in every image. The implementation of *noScale* is also responsible to produce a follow-up test case. Since the simplification percentage is an integer in the input 3D polygonal model, the implementation always assigns 100 to this integer.

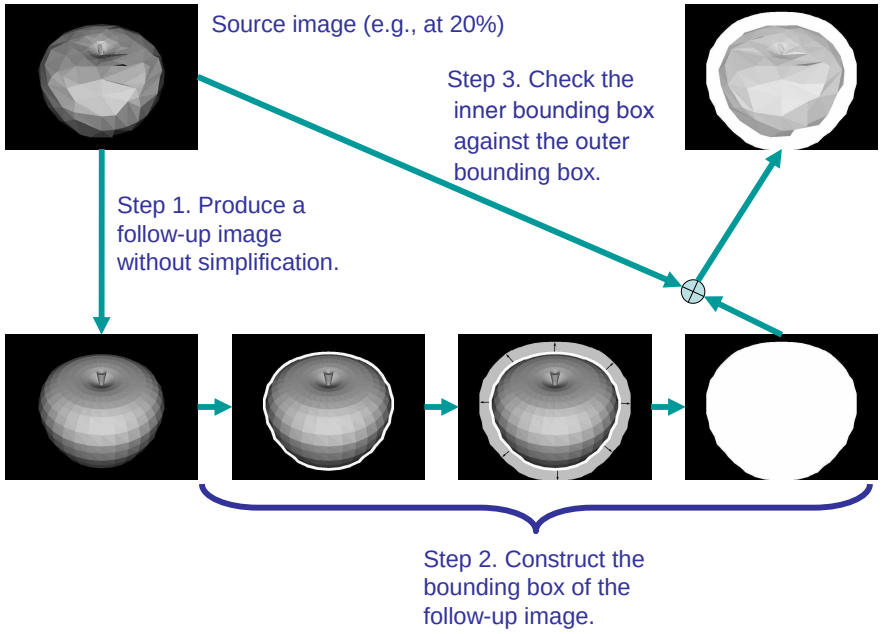


Figure 4: An Illustration of MR_1

[6] Note that we compare the outlines of shapes rather than comparing the outline against a shape. We illustrate the inner apply object rather than the outline of the inner apple in the figure to ease readers to spot the comparison between outlines.

- The second metamorphic relation (MR_2) reverses the order of the vertices of a given polygonal model to produce the reversed sequence of the polygonal model. It further checks whether the image outputs of the program over two polygonal models are the same. It is analogous to requesting the program to visualize the same graphics using different sequences of operations. We depict MR_2 in Figure 5.

  A 3D polygonal model in a PLY file is a sequence of vertices $\langle v_1, v_2, \ldots, v_{n-1}, v_n \rangle$. Let *reverse* be a standard sequence reversal function[7] that accepts a sequence $\langle v_1, v_2, \ldots, v_{n-1}, v_n \rangle$, reverses the order of the elements in the specified sequence, and returns the reversed sequence $\langle v_n, v_{n-1}, \ldots, v_2, v_1 \rangle$. Let m be a test case. The metamorphic relation is as follows:

  $$P(m) = P(reverse(m)).$$

  The construction of the follow-up test case is not difficult. In the implementation, we extract the sequence from a source PLY file, and then reverse the sequence. We further replace the original sequence in the source PLY file by the reversed sequence to produce another PLY file (i.e., the follow-up test case). The simplification percentage of the follow-up test case is set to be the same as that of the given polygonal model. To implement the equality of the metamorphic relation, we compare the images via vectors of extracted features directly.

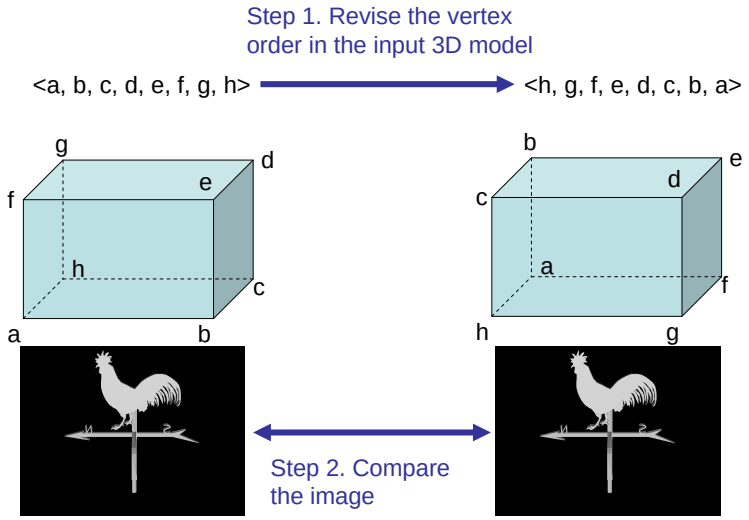


Figure 5: An Illustration of MR_2

- The third metamorphic relation (MR_3) changes the input so that the image output should be upside down. This is similar to the "flip vertical" function in the drawing toolbar of Microsoft Office. Then, it further changes the image so that it is upside down again. The net result should be as if no any flip operation were applied to the inputs and the outputs. We depict MR_3 in Figure 6.

[7] Such as "static void java.util.Collections.reverse(List list)".

The implementation of this metamorphic relation is also not hard. Each vertex q in a 3D polygon model is a 3D coordinate ($coor_x$, $coor_y$, $coor_z$), where each coordinate component is a number. Thus, for the sequence of vertices representing the 3D polygon model, we simply compute a new y-coordinate of every vertex in the sequence by the Equation (1) to form ($coor_x$, $coor'_y$, $coor_z$):

$$coor'_y = -coor_y. \tag{1}$$

It produces a follow-up 3D polygonal model that will turn the original 3D polygonal model upside down. To compare the image output of the original test case and that of the follow-up test case, the implementation runs Photoshop 7.0 to invert the image output of the follow-up test case. It then compares this inverted image directly with the image output of the initial test case via vector of extracted features. Like the implementation of the second metamorphic relation, the simplification percentage of the follow-up polygonal model is the same as that of the initial test case.

Formally, let *yInvert* be a function that accepts a 3D polygonal model, performs the y-coordinate transformation stated in Equation (1) over the sequences of vertices in the model, and returns a 3D polygonal model that its z-coordinates has transformed. Let *flip* be a function to invert an input image. Let m be an initial test case. The metamorphic relation is as follows:

$$P(m) = flip(P(yInvert(m))).$$

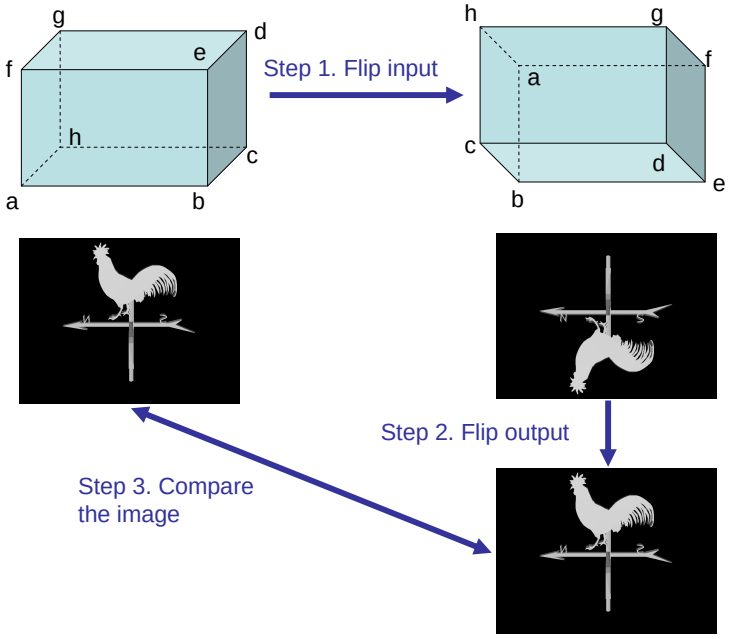


Figure 6: An Illustration of MR_3

Let us firstly estimate the overall capability of failure identification by these MRs. Because of the sheer size of the test pool used in the experiments, we cannot do the estimation by applying the MRs exhaustively to all the test cases. We randomly select a subset as initial test cases. Using our implementation for exercising MT, we construct follow-up test cases and then check whether they reveal a failure. The results show that 29.4% of the failure-causing test cases for *Melax*, 34.1% for *Quadric*, and 36.3% for *QuadricTri* are detected.

Readers may express a concern that these metamorphic relations appear to be weak. We thus refine **RQ1** to a more tractable research question below:

> **RQ1$^+$:** *Can the effectiveness of failure identification be improved by piping results of a pattern classification approach to weak MRs?*

5.2 Experimental Procedure

In this section, we describe the experimental procedure to evaluate PAT 2. First, the classification result of PAT 1.r and PAT 1.d are directly obtained from the dataset of the previous experiment [12] (see Section 5.1.2). To produce the test results of PAT 1.0, we group the classification result of PAT 1.r and that of PAT 1.d to denote the result of PAT 1.0. They serve as the baseline for evaluating the performance of PAT 2.0.

To produce the test results of PAT 2.r for a particular IUT, we first use the passed test cases of PAT 1.r for that IUT, and apply every such passed test case as an initial test case of each metamorphic relation. The MT module uses each metamorphic relation implementation to construct a follow-up test case based on the initial test case. Then, PAT 1.r determines whether the follow-up test case has passed or is failure-causing. If the test verdict of the follow-up test case has been marked as passed, the MT module will continue to compare the initial test case, the follow-up test case, and their outputs based on the metamorphic relation implementation that constructs the follow-up test case. If the MT module reveals a failure, we erase the previous labels of the initial test case and the follow-up test case, and mark them as failure-causing instead. We iterate the same procedure for every subject program. We further repeat the same procedure for PAT 2.d by using the passed test case of PAT 1.d as the initial test cases for the MT module. Similarly to PAT 1.0, we group the classification result of PAT 2.r and that of PAT 2.d to denote the result of PAT 2.0.

In summary, the variables of the controlled experiment are as follows:

Independent Variables. There are four independent variables. They are the subject programs to train a classifier, the subject programs used as the IUT, using MT or not, and the number of input models to train a classifier.

Dependent Variables. There is only one dependent variable, which is the label of each test case.

Control Variables. There are a number of control variables. They are the implementation languages to implement the subject programs (Java in our case), the chosen classifier (the C4.5 classifier), the metamorphic relations (the three relations stated in Section 5.1.4), the polygonal models to construct the test case pool (Θ), the features used for classification (the

black-box features), the way to construct faulty versions to train a classifier and simulate faulty behavior in the program under test (the program mutation approach using *mu*Java).

5.3 Empirical Results and Analysis

In this section, we present the results of the empirical study.

5.3.1 Investigations on Effectiveness Improvement

In this section, we examine whether the introduction of an MT module improve the effectiveness of PAT 1.

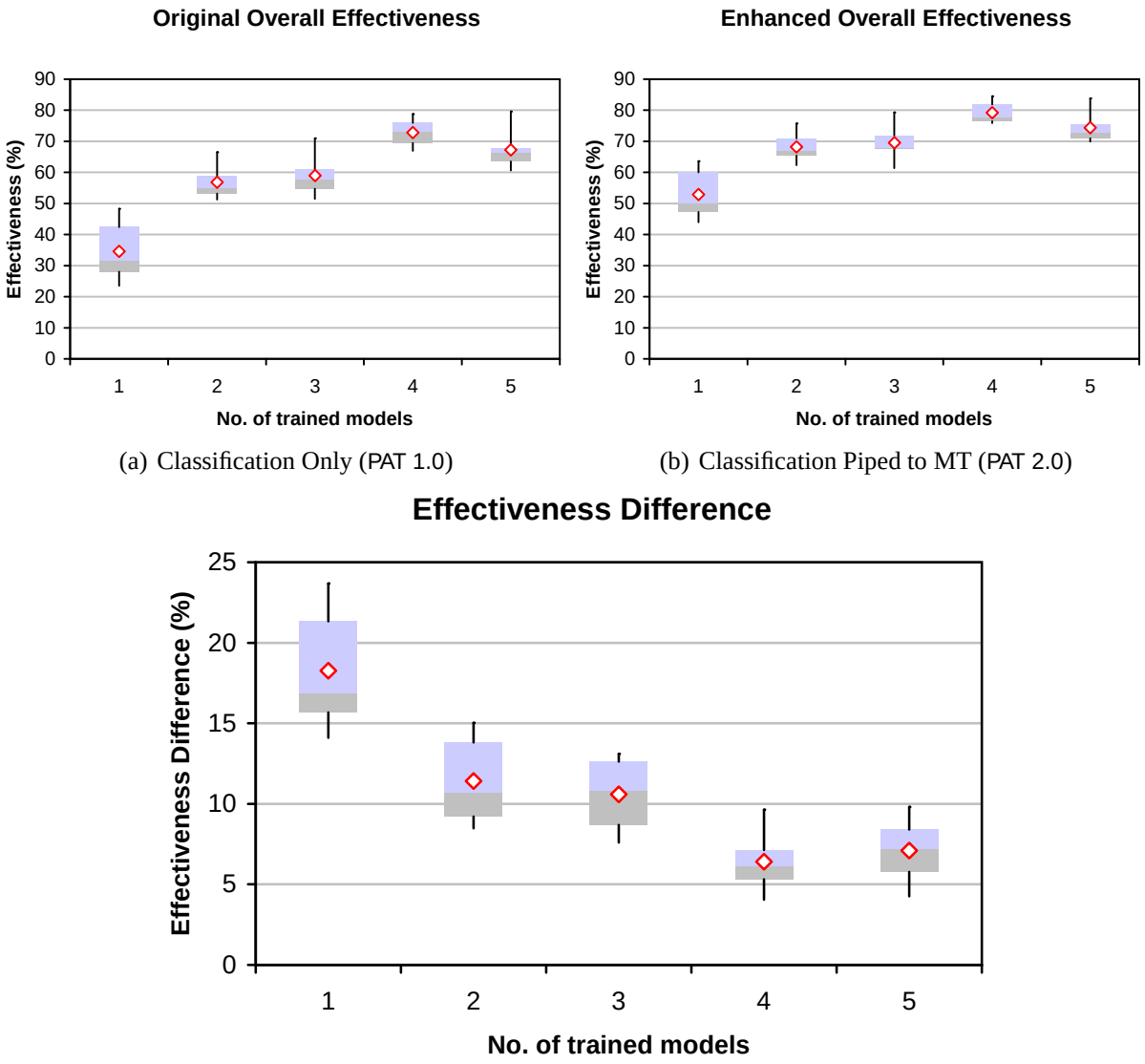
Figures 7(a) and (b) present box-and-whisker plots of the effectiveness of PAT 1.0 and PAT 2.0, respectively. The *y*-axis shows the effectiveness and the *x*-axis shows the number of 3D polygonal models applied to train the C4.5 classifier during the PAT 1.0 phases. Unless specified, in the rest of the paper, readers can interpret the pairs of axes in each plot similar to the plot in Figure 7(a). Figure 7(c) shows the difference in effectiveness between PAT 2.0 and PAT 1.0 by the formula: Effectiveness of PAT 2.0 $-$ Effectiveness of PAT 1.0.

We observe from the three plots that MT improves the effectiveness of the classification module. This is a direct consequence of our new methodology because the main difference between PAT 1.0 and PAT 2.0 is the presence of an MT module, which constructs follow-up test cases from the passed initial test cases to identify potential additional failures. The trends in the plots show that the marginal improvement in the effectiveness for PAT 1.0 decreases as the number of input 3D polygonal models used in the training phase of the classification module (PAT 1.0) increases. This is understandable: As the effectiveness improves, the room for further improvement reduces.

To study the improvement statistically, we apply the two-sided Mann-Whitney U test (denoted by U-test) on the effectiveness values of the classification approach with and without the MT module (i.e., comparing the effectivenesses of PAT 1.0 and PAT 2.0). The results are z-score = 3.060 and p-value = 0.0022 (< 0.05). We further apply the Wilcoxon Matched-Pairs Signed-Rank Test (rank-test) to check the hypothesis, giving: p-value ≈ 0.0000 (< 0.05). Therefore, we reject hypothesis H_1 at the 5% significance level, and conclude that piping to the MT module *does* improve the effectiveness of the classification approach significantly.

Next, we continue to study the impact of using MT over PAT 1.r and over PAT 1.d. In other words, we compare PAT 2.r with PAT 2.d. Figure 8 shows the trends of effectiveness of PAT 2.r and PAT 2.d, respectively.

Each plot in Figure 8 shows that having an MT module improves the effectiveness of the approach without the MT module. We also perform the U-test and rank-test on them. For the comparison between PAT 1.r and PAT 2.r, U-test shows that z-score = 1.663 p-value = 0.0963 ($<$0.10), while rank-test shows that p-value = 0.0020 (< 0.10). On one hand, we reject hypothesis H_2 at the 10% significance level. On the other hand, the results of the hypothesis test reveal that improvements by the MT module are observable, but not highly significant. At first sight, it may be relevant to the use of weak metamorphic relations.



(a) Classification Only (PAT 1.0)

(b) Classification Piped to MT (PAT 2.0)

(c) Difference between (b) and (a)

Figure 7: Changes in Overall Effectiveness by the Use of MT

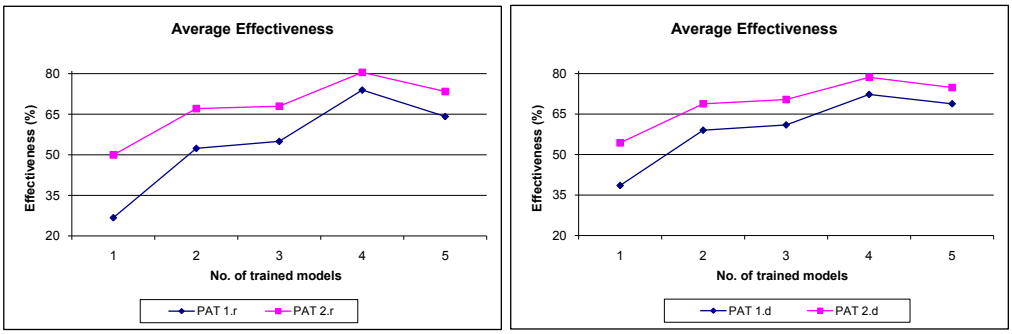
However, a comparison between PAT 1.d and PAT 2.d shows that, for U-test, z-score = 2.408 and p-value = 0.0161 (< 0.05), while for rank-test, p-value = 1.91e-06 (< 0.05). On one hand, we reject hypothesis H_3 at the 5% significance level. On the other hand, the metamorphic relations are already strong enough to make a significant difference between PAT 1.d and PAT 2.d.

The above hypothesis testing results show that:

(1) If we do not distinguish the types of reference model to train a classifier, MT provides a significant improvement (thus, rejecting H_1). Intuitively, this finding relieves testers from the worry that a resembling reference model may be mistaken to be dissimilar, and makes MT to

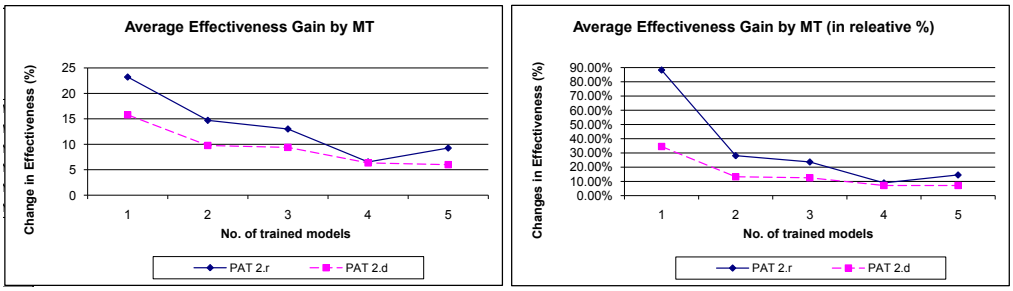
be a good complement to a classification approach. However, as we will discuss in the next section, it is not the case when we also consider robustness of the classification scheme.

(2) If a tester heeds the advice of [10] and uses a resembling reference model to train up a classifier, the advantages of piping the results of the classification module to an MT module may be not traditionally significant. The experiment result does not support us to reject H_2 at a high, say 5%, significance level. On the other hand, the difference is noticeable (see Figure 8(a)). Indeed, the finding supports us to reject H_2 at a good (10%) significance level. Viewing the results from another perspective, it may already show that PAT 1's advice is useful, and PAT 2 is a healthy option for testers to improve the effectiveness of identifying failures in testing their programs. We will further study this point in Section 5.3.2.



(a) PAT 1.r vs. PAT 2.r (b) PAT 1.d vs. PAT 2.d

Figure 8: Effectiveness of PAT 2.r over PAT 1.r, and PAT 2.d over PAT 1.d



(a) Changes in (Absolute) Effectiveness (b) Changes in (Relative) Effectiveness

Figure 9: Comparing the Changes in Effectiveness of PAT 2.r over PAT 1.r and PAT 2.d over PAT 1.d

Figure 9 compares the differences in effectiveness of PAT 2.r over PAT 1.r to that of PAT 2.d over PAT 1.d. We want to study whether the effectiveness improvement of MT added to PAT 1.r is the same as that added to PAT 1.d. We calculate the change in effectiveness between PAT 2.r over PAT 1.r by the formula (F1): Effectiveness of PAT 2.r − Effectiveness of PAT 1.r. For comparison in relative terms, we divide the result of formula (F1) by the effectiveness of PAT 1.r.

Similarly, we calculate the change in effectiveness between PAT 2.d over PAT 1.d by the formula (F2): Effectiveness of PAT 2.d − Effectiveness of PAT 1.d. For comparison in relative terms, we divide the result of formula (F2) by the effectiveness of PAT 1.d.

Figure 9 shows the average effectiveness improvements of PAT 2.r and PAT 2.d (over PAT 1.r and PAT 1.d, respectively) in both absolute and relative terms. We observe that the solid line is always higher than the dotted line in either plot. It indicates that the average improvements of PAT 2.r are always more than those of PAT 2.d, which may indicate that the former type of improvement is better than the latter type.

Since we have observed that MT has a positive effect on effectiveness, we further conduct a one-tailed U-test to compare the two types of improvement. We find that, for improvements in the absolute terms, the two distributions are different, where z-score = 1.716 and p-value = 0.0431 ($<$0.05). Similarly, for relative improvements, the two distributions are also different, where z-score = 1.408 and p-value = 0.0796 ($<$0.10). The findings support us to reject hypothesis H_4 at the 10% significance level. It indicates that the improvement of using a resembling reference model is better than that using a dissimilar reference model. We will further study the issue in the next section and in Section 6.

We note that rank-test requires the lists to have the same number of elements. This is the case when comparing an approach with the same approach enhanced by MT. On the other hand, among the three subject programs in our empirical study, PAT 1.r use the pair *Quadric* and *QuadricTri* because they resemble each other. PAT 1.d uses other two pairs of subject programs, namely *Quadric* and *Melax*, and *QuadricTri* and *Melax*. Consequently, we do not apply rank-test to hypothesis H_4. Table 4 summarizes the hypothesis testing results presented in this section.

Hypothesis	Brief Description	Result	Sign. Level
H_1	PAT 1.0 = PAT 2.0 ?	rejected	5%
H_2	PAT 1.r = PAT 2.r ?	rejected	10%
H_3	PAT 1.d = PAT 2.d ?	rejected	5%
H_4	(PAT 2.r − PAT 1.r) = (PAT 2.d − PAT 1.d) ?	rejected	10%

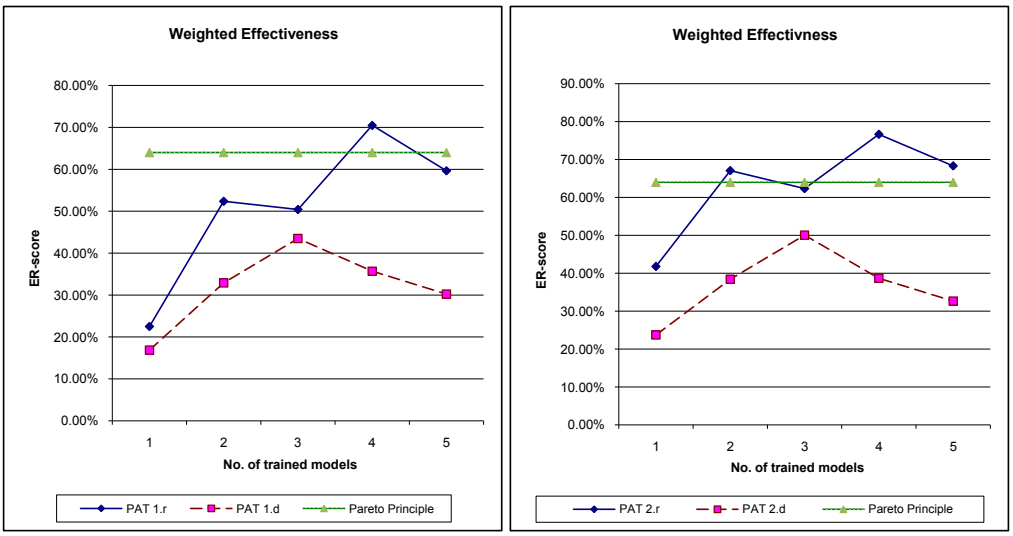
Table 4: Summary of Analysis of the Use of an MT module on Top of a Classification Approach (Effectiveness)

In the next section, we shall consider the issue of robustness and re-examine our findings.

5.3.2 Taking Robustness into Account

In this section, we further consider both effectiveness and robustness when evaluating a testing technique that involves pattern classification. Just for the sake of argument, a classifier may always label all test cases as passed, meaning that it will never raise a false-positive case and keep all potential failed test cases as false negatives. This is undesirable from the testing point of view because the sole purpose to train the classifier is to identify failures from program outputs. Similarly, the other extreme situation is to force a classifier to always label a test case as failed. It always reveals failures (with potentially many false-positive cases).

Recall from Section 5.1.3 that *robustness* is the percentage of the number of false positives to the sum of the numbers of false positives and true negatives. MT requires multiple test cases to be checked against a metamorphic relation. As we will explain in Section 6, MT may mark a passed test case as failure-causing. In other words, it may move test cases from the true positive category to the false negative category (but not vice versa). The net result will affect the robustness of the testing technique. To take robustness into consideration, we use the ER-score as the measure.



(a) Comparing PAT 1.r and PAT 1.d (b) Comparing PAT 2.r and PAT 2.d

Figure 10: Comparing the ER-score of PAT 1.r and PAT 1.d with and without MT

Figures 10(a) and (b) show observable differences between PAT 1.r and PAT 1.d, and between PAT 2.r and PAT 2.d, respectively. The *y*-axis of each plot is the ER-score, and the *x*-axis is same as that in Figure 7(a). We follow the same hypothesis testing technique as presented in Section 5.3 and find that, for the comparison the differences in ER-scores between PAT 2.r over PAT 1.r and PAT 2.d over PAT 1.d, U-test yields z-score = 3.696 and p-value = 0.0002 (< 0.05). We thus reject hypothesis H_4 at the 5% significance level.

Similarly, for the comparison between PAT 1.r and PAT 2.r, U-test gives z-score = 1.890 and p-value = 0.0588 (≈ 0.05) while rank-test gives p-value = 0.0020 (< 0.05). We conclude that we may marginally reject H_2 at the 5% significance level. We note that in statistics, there is no firm rule to draw a fine line between a level of significance and the otherwise. To be conservative about the analysis, we also conclude that H_2 is rejected at the 10% significance level. Nonetheless, when we conduct hypothesis testing on H_1 and H_3, our findings do not support us to reject either hypothesis even at the 10% significance level for either U-test or rank-test.

We have computed the average improvement in ER-score from PAT 1.r to PAT 2.r, and that from PAT 1.d to PAT 2.d; the results are 12.16% and 4.86%, respectively. The findings further indicate that the improvement on the use of resembling reference models is more significant than that on the use of dissimilar reference models.

In Figure 10, we also show a horizontal line labeled as the Pareto Principle (also known as the 80-20 rule) [55]. Since we have two dimensions (effectiveness and robustness) in the ER-score, in order to apply this principle, we should have a line drawn at 64%, which means 80% in effectiveness and 80% in robustness ($0.8 \times 0.8 = 0.64$). Figure 10(b) shows that, with the improvement on PAT 1.r by PAT 2.r, constructing a pseudo-oracle that exceeds (or is close) to the threshold defined by the Pareto Principle is possible. On the contrary, Figure 10(a) shows that such a threshold is less attainable by a classification approach without the MT module, or PAT 2.d.

Thus, if we interpret the Pareto Principle as a practical guideline to offer a solution to practitioners, PAT 2.r is more useful than PAT 2.d. We also summarize the analysis result in Table 5.

Hypothesis	Brief Description	Result	Sign. Level
H_1	PAT 1.0 = PAT 2.0 ?	accept	10%
H_2	PAT 1.r = PAT 2.r ?	rejected	10%
H_3	PAT 1.d = PAT 2.d ?	accept	10%
H_4	(PAT 2.r − PAT 1.r) = (PAT 2.d − PAT 1.d) ?	rejected	5%

Table 5: Summary of Analysis of the Use of an MT module on Top of a Classification Approach (ER-score)

Table 4 and Table 5 help us answer the research question **RQ1**$^+$: When resembling reference model is used, in both effectiveness and ER-score, the improvement of having an MT module is significant. With the comparison to the Pareto Principle, this advice can be practical.

5.4 Threats to Validity

In this section, we discuss the threats to validity of our empirical study.

Internal validity concerns whether our findings truly represent a cause-and-effect relationship that follows logically from the design and execution of our experiment. The choice of the three metamorphic relations is based on our experience. We have implemented the metamorphic relations in Java in general, and used a commercial tool (Photoshop) to process the images. As we have described how we implement them in Section 5.1.4, all parts of their implementations are simple. We have conducted code inspection and run a few tests to assure the quality of these implementations. Photoshop is a popular product, and its perceived output quality is reliable. In the experiment, we compare the results against the baseline results provided by the C4.5 classifier. C4.5 is a classic classifier and has been widely used in machine learning, data mining, and visualization research. The tool (WEKA) that implements the C4.5 classifier is also widely used in research studies. We have surveyed over the Internet about the problems of using WEKA. We are not aware of any reported problem about the accuracy of the C4.5 implementation.

External validity concerns the applicability and generality of our results. The quality of metamorphic relations can be important to reveal failures. We have deliberately used simple metamorphic relations in our study. The results of our experiments serve as a baseline for further investigations. OpenGL is used in the implementations of the subject programs to visualize graphics. While OpenGL is a popular standard, there are other choices such as DirectX and

Flash. It is interesting to know whether different implementation languages would have significant differences in testing effectiveness. We have only experimented with a few implementations of mesh simplification algorithms. There are many other visualization algorithms. The generalization of our proposal, therefore, warrants more research. Also, our work is built on top of the C4.5 classifier. While it is an important and classical algorithm in data mining, using other classifiers may give different results. This is thus interesting to know the results of using the other classifiers in the future. Our experiment uses a set of 44 open-source 3D polygonal models to create test cases. They include a portrait of Beethoven, a chair, a spider, a teapot, a tennis shoe, a weathervane, a street lamp, a sandal, a cow, a Porsche car, an airplane, and so on. The collection includes many different graphics of diverse shapes and many representative geometric appearances. Some of the polygonal models such as the portrait of Beethoven have been widely used in graphics research. We have done our best to conduct the experiment. We have used 10 machines at our student laboratory to execute the experiment for more than two consecutive months. We believe that it simulates the practical testing effort in real life.

Because of the test oracle problem in verifying graphical outputs, we have used feature extraction techniques to tackle the issue instead of directly comparing the actual outputs in PAT 1. We realize from the machine learning community that feature selection plays a central role in the effectiveness of a classifier. Intuitively, using a different set of features in an experiment to train a classifier and use the trained classifier to reveal failures may affect the result. This will affect whether an initial test case of our metamorphic testing phase has been classified correctly. Therefore, it affects both the effectiveness and ER-score of the experiment. To ease this threat, we use generic features such as the standard frequency spectrum in the experiment to evaluate PAT 1.

To produce failed test results to train a classifier, we have used mutation analysis in general and the mutants generated by muJava in particular. Andrews et al. [2] find that the use of mutation operators can yield trustworthy results for test experiments. Kapoor [31] proves that the coupling hypothesis of mutation testing holds in many classes of logical fault, and further extend the fault class hierarchy for logical faults of Lau and Yu [35] in his work with Bowen [32]. On the other hand, developers may produce other realistic faults in a program. Apart from using mutation analysis, therefore, one potential way to complement our methodology is to extract the faults from the repository of mesh simplification programs and simulate them as faulty versions of a reference model. We leave the evaluation of the feasibility of such a strategy as future work.

Construct validity seeks agreement between our intent of measure and the procedures of our measurement used in the experiment. We are dealing with visualization-intensive software in our study. The way to sample frequencies from the image outputs and summarize them into vectors of extracted features may affect the results. As we have described in the setup of the experiment, many frequency values may be extracted from an image. We sample at the mean plus/minus one to three standard deviations to avoid biases toward particular ranges of frequency values. We use effectiveness and ER-score to measure the actual results of the test experiment. Effectiveness is defined as sensitivity, which is widely used in measuring the performance of a binary classification scheme. The ER-score combines sensitivity and specificity into one value. We intend to measure how the effectiveness of our approach may be affected when specificity varies. We have designed the ER-score with care so that it does not favor either sensitivity or

		Expected outcome	
		TRUE	FALSE
Test outcome	Positive	True positive (↑)	False Positive (↑)
	Negative	Fase Negative (↓)	True Negative (↓)

Table 6: The Impact of PAT 2 over PAT 1

specificity, and varies proportionally to either sensitivity or specificity when the corresponding counterpart is kept as a constant.

6 Discussion

In this section, we further discuss the findings obtained in Sections 5.3.1 and 5.3.2. We generally look at the impact of MT from the perspective of effectiveness and ER-score over a binary classification scheme. Finally, we will examine the resulting conclusions of the hypothesis tests.

Our basic idea in PAT 2 is to apply MT on test cases marked as passed by PAT 1 and, if MT reveals a failure, re-label a passed test case as failure-causing. As we have described in Section 5.1.3, a passed test case of a classifier (produced by PAT 1) may be true negative (TN) or false negative (FN). In other words, an initial test case of the MT phase may be true negative or false negative, and a follow-up test case of the MT phase alike. A pair of passed initial test case and passed follow-up test case may thus fall within one of the four possible combinations, namely, TN-TN, FN-FN, TN-FN, and FN-TN.

Three of the above four combinations involve at least one test case that is false negative. For each of these three combinations, the MT phase may thus re-label the test cases as failure-causing. It means that MT may move a test case from the false negative category to the true positive category in our binary classification scheme. At the same time, because MT identifies a failure through a relation over test cases (rather than through one particular test case), MT may mark a passed test case as failure-causing, which means that MT may also move a test case from the true negative category to the false positive category.

Table 6 summarizes the changes in the classification category based on the above analysis on the impact of MT over a binary classification scheme. In Table 6, each of the four categories shows an arrow pointing either upwards or downwards. It depicts the direction of impact of MT over PAT 1. An upward (downward) arrow indicates that MT adds (removes) test cases to (from) the category.

Therefore, according to the specificity formula presented in Section 5.1.3, MT may improve (but not worsen) the sensitivity of a binary classification scheme, and worsen (but not improve) the specificity of the same scheme. In statistics, Type I error and Type II error describe possible

errors made in a statistical decision process. Type I error refers to the error of rejecting a correct null hypothesis, while Type II error refers to the error of not rejecting a false null hypothesis. In essence, PAT 2 trades Type 1 error of PAT 1 for Type II error of PAT 1 through the application of MT.

To examine such tradeoff, we have used the ER-score as the measure in the experiment presented in this paper. However, because of tester preference, we have separately studied the effectiveness of the testing technique.

We have rejected both hypotheses H_1 and H_3 in Section 5.3.1 when we measure effectiveness only; and yet, we have failed to do them again when using ER-score as the metric to measure the above-mentioned tradeoff in Section 5.3.2. The combined results show that the effectiveness of PAT 1 does improve significantly through the application of MT, yet the Type 1 error has also increased to the extent that prevents significant improvement to be claimed. In Figure 10(b), we have however observed noticeable improvement of PAT 2.d over PAT 1.d. They show that the improvement on effectiveness (sensitivity) versus the deterioration in robustness (specificity) produced by MT is asymmetric but not significantly different. We believe that there are types of MT that can be symmetric or significantly asymmetric in the sensitivity-specificity tradeoff perspective. It may be worth studying the types of MT to study the tradeoff more comprehensively in the future.

Accepting H_1 in the tradeoff analysis shows that testers may not blindly apply an arbitrary reference model to obtain a significantly better result from PAT 2 over PAT 1. In practice, it means that testers need spending efforts to confirm whether a reference model resembles the implementation under test. We tend to believe that, in practice, the developers of the implementation under test can provide such expert judgment.

Nonetheless, not every version of PAT 2 can be practical. We have compared our results with the Pareto Principle in the tradeoff analysis (see Figures 10(a) or (b)). We have found that both PAT 1.d and PAT 2.d are less effective than the threshold lines that represent the Pareto Principle. To use PAT 2 effectively, again, the identification of resembling reference models is important.

We have further rejected H_2 and H_4 in both Section 5.3.1 and Section 5.3.2 at the 10% significance level successfully. We have however argued in these two sections that typically, rejecting a hypothesis test in statistics would require the significance setting at the 5% level. Nonetheless, we believe that having results established at the 10% significance level represents a promising effect.

We have assessed that PAT 2.r can attain the threshold level of Pareto Principle. We believe that PAT 2.r is thus more accessible to the practitioners than the other versions that we have studied in the empirical study.

7 Conclusion

Mesh simplification is a technique to create graphics at different levels of details. It simplifies a three-dimensional (3D) polygonal model to the one with fewer polygons and aims to preserve the appearances of the original model as much as possible. Different such techniques, however,

optimize different perspectives such as speed or graphical shadow of the shape. As a result, they produce different graphics, although the graphics look coarsely similar in appearance. Defining the expected results of test cases is thus hard, which causes a test oracle problem when testing mesh simplification programs. Our previous work recognizes the use of resembling reference models to guide the training phase. Still, owing to the statistical nature of classifiers, many test cases classified into the passed category may, in fact, be failure-causing, thus lowering the testing effectiveness in identifying failures.

In this paper, we have proposed an integrated approach that pipes the test results from a pattern classification module to a metamorphic testing (MT) module for follow-up testing. Specifically, it uses the metamorphic testing approach to check the test results marked as passed by a classifier. We have reported an empirical study that applies three simple and general metamorphic relations to produce follow-up test cases to evaluate our proposal. For effectiveness, the integrated approach significantly improves the pure pattern classification approach. When we consider robustness as well, the integrated approach using a resembling reference model gives significantly better improvement over the one using a dissimilar reference model. We have also explained why and how MT in our methodology represents a technique to trade specificity for sensitivity.

Our proposal has showed a strategy that aligns a statistical approach with an analytical approach to give better results. We believe that such a strategy in general has applicable scenarios in other application domains to gain fruit results. Future work includes new techniques to filter out false-positive cases in the failed category, a tighter integration of pattern classification and metamorphic testing, and an underpinning theory. We have not studied real-time testing issues in assuring mesh simplification programs. We will study these issues in the future.

Acknowledgements

We would like to thank the anonymous reviewers for their time and constructive comments, and Prof. Jeff Offutt for the discussion on the representativeness of mutation analysis to real faults.

References

[1] M.N. Ahmed, S.M. Yamany, N. Mohamed, A.A. Farag, and T. Moriarty. A modified fuzzy c-means algorithm for bias field estimation and segmentation of MRI data. *IEEE Transactions on Medical Imaging*, 21 (3): 193–199, 2002.

[2] J.H. Andrews, L.C. Briand, and Y. Labiche. Is mutation an appropriate tool for testing experiments? In *Proceedings of the 27th International Conference on Software Engineering* (*ICSE 2005*), pages 402–411. ACM, New York, NY, 2005.

[3] L. Baresi, G. Denaro, L. Mainetti, and P. Paolini. Assertions to better specify the Amazon bug. In *Proceedings of the 14th International Conference on Software Engineering and Knowledge Engineering* (*SEKE 2002*), pages 585–592. ACM, New York, NY, 2002.

[4] J. Berstel, S.C. Reghizzi, G. Roussel, and P. San Pietro. A scalable formal method for design and automatic checking of user interfaces. *ACM Transactions on Software Engineering and Methodology*, 14 (2): 124–167, 2005.

[5] A. Bierbaum, P. Hartling, and C. Cruz-Neira. Automated testing of virtual reality application interfaces. In *Proceedings of the Eurographics Workshop on Virtual Environments* (*EGVE 2003*), pages 107–114. ACM, New York, NY, 2003.

[6] R.V. Binder. *Testing Object-Oriented Systems: Models, Patterns, and Tools*. Addison Wesley, Reading, MA, 2000.

[7] J.F. Bowring, J.M. Rehg, and M.J. Harrold. Active learning for automatic classification of software behavior. In *Proceedings of the 2004 ACM SIGSOFT International Symposium on Software Testing and Analysis* (*ISSTA 2004*), *ACM SIGSOFT Software Engineering Notes*, 29 (4): 195–205, 2004.

[8] L.C. Briand, M. Di Penta, and Y. Labiche. Assessing and improving state-based class testing: a series of experiments. *IEEE Transactions on Software Engineering*, 30 (11): 770–783, 2004.

[9] W.K. Chan, T.Y. Chen, H. Lu, T.H. Tse, and S.S. Yau. Integration testing of context-sensitive middleware-based applications: a metamorphic approach. *International Journal of Software Engineering and Knowledge Engineering*, 16 (5): 677–703, 2006.

[10] W.K. Chan, M.Y. Cheng, S.C. Cheung, and T.H. Tse. Automatic goal-oriented classification of failure behaviors for testing XML-based multimedia software applications: an experimental case study. *Journal of Systems and Software*, 79 (5): 602–612, 2006.

[11] W.K. Chan, S.C. Cheung, J.C.F. Ho, and T.H. Tse. Reference models and automatic oracles for the testing of mesh simplification software for graphics rendering. In *Proceedings of the 30th Annual International Computer Software and Applications Conference* (*COMPSAC 2006*), volume 1, pages 429–438. IEEE Computer Society, Los Alamitos, CA, 2006.

[12] W.K. Chan, S.C. Cheung, J.C.F. Ho, and T.H. Tse. PAT: a pattern classification approach to automatic reference oracles for the testing of mesh simplification programs. *Journal of Systems and Software*, 2008. doi.10.1016/j.jss.2008.07.019.

[13] W.K. Chan, J.C.F. Ho, and T.H. Tse. Piping classification to metamorphic testing: an empirical study towards better effectiveness for the identification of failures in mesh simplification programs. In *Proceedings of the 31st Annual International Computer Software and Applications Conference* (*COMPSAC 2007*), volume 1, pages 397–404. IEEE Computer Society, Los Alamitos, CA, 2007.

[14] 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.

[15] S.C. Cheung, S.T. Chanson, and Z. Xu. Applying generic timing tests for distributed multimedia software systems. *IEEE Transactions on Reliability*, 53 (3): 329–341, 2004.

[16] P. Cignoni, C. Rocchini, and G. Impoco. A comparison of mesh simplification algorithms. *Computers and Graphics*, 22 (1): 37–54, 1998.

[17] R.L. Cook, J. Halstead, M. Planck, and D. Ryu. Stochastic simplification of aggregate detail. *ACM Transactions on Graphics*, 26 (3): Article No. 79, 2007.

[18] B. d'Ausbourg, C. Seguin, G. Durrieu, and P. Roch. Helping the automated validation process of user interfaces systems. In *Proceedings of the 20th International Conference on Software Engineering* (*ICSE 1998*), pages 219–228. IEEE Computer Society, Los Alamitos, CA, 1998.

[19] C. DeCoro and N. Tatarchuk. Real-time mesh simplification using the GPU. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*, pages 161–166. ACM, New York, NY, 2007.

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

[21] L.K. Dillon and Y.S. Ramakrishna. Generating oracles from your favorite temporal logic specifications. In *Proceedings of the 4th ACM SIGSOFT Symposium on Foundations of Software Engineering* (*SIGSOFT '96/FSE-4*), *ACM SIGSOFT Software Engineering Notes*, 21 (6): 106–117, 1996.

[22] R.O. Duda, P.E. Hart, and D.G. Stork. *Pattern Classification*. Wiley, New York, NY, 2000.

[23] C. Fahn, H. Chen, and Y. Shiau. Polygonal mesh simplification with face color and boundary edge preservation using quadric error metric. In *Proceedings of the 4th IEEE International Symposium on Multimedia Software Engineering* (*MSE 2002*), pages 174–181. IEEE Computer Society, Los Alamitos, CA, 2002.

[24] P. Francis, D. Leon, M. Minch, and A. Podgurski. Tree-based methods for classifying software failures. In *Proceedings of the 15th International Symposium on Software Reliability Engineering* (*ISSRE 2004*), pages 451–462. IEEE Computer Society, Los Alamitos, CA, 2004.

[25] M. Garland and P. Heckbert. Surface simplification using quadric error metrics. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques* (*SIGGRAPH 1997*), pages 209–216. ACM, New York, NY, 1997.

[26] R.B. Grinde, and T.M. Cavalier. A new algorithm for the two-polygon containment problem. *Computers and Operations Research*, 24 (3):231–251, 1997.

[27] R.C. Gonzalez and R.E. Woods. *Digital Image Processing*. Prentice Hall, Englewood Cliffs, NJ, 2002.

[28] R.M. Hierons. Avoiding coincidental correctness in boundary value analysis. *ACM Transactions on Software Engineering and Methodology*, 15 (3):227–241.

[29] D.W. Hosmer and S. Lemeshow. *Applied Logistic Regression*. Wiley, New York, NY, 2004.

[30] P. Hu, Z. Zhang, W.K. Chan, and T.H. Tse. An empirical comparison between direct and indirect test result checking approaches. In *Proceedings of the Third International Workshop on Software Quality Assurance* (*SOQUA 2006*) (*in conjunction with the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (*SIGSOFT 2006/FSE-14*)), pages 6–13. ACM, New York, NY, 2006.

[31] K. Kapoor. Formal analysis of coupling hypothesis for logical faults. *Innovations in Systems and Software Engineering*, 2 (2): 80–87, 2006.

[32] K. Kapoor and J.P. Bowen. Test conditions for fault classes in Boolean specifications. *ACM Transactions on Software Engineering and Methodology*, 16 (3): 1–12, 2007.

[33] R. Kohavi and F. Provost. Glossary of terms. *Machine Learning*, 30 (2/3): 271–274, 1998.

[34] M. Last, M. Friedman, and A. Kandel. The data mining approach to automated software testing. In *Proceedings of the 9th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining* (*KDD 2003*), pages 388–396. ACM, New York, NY, 2003.

[35] M.F. Lau and Y.T. Yu. An extended fault class hierarchy for specification-based testing. *ACM Transactions on Software Engineering and Methodology*, 14 (3): 247–276, 2005.

[36] P. Lindstrom and G. Turk. Imagine-driven simplification. *ACM Transactions on Graphics*, 19 (3): 204–241, 2000.

[37] D.P. Luebke. A developer's survey of polygonal simplification algorithms. *IEEE Computer Graphics and Applications*, 21 (3): 24–35, 2001.

[38] D.P. Luebke, M. Reddy, J.D. Cohen, A. Varshney, B. Watson, and R. Huebner. *Level of Detail for 3D Graphics*. Morgan Kaufmann, San Francisco, CA, 2003.

[39] Y.-S. Ma, A.J. Offutt, and Y.-R. Kwon. MuJava: an automated class mutation system. *Software Testing, Verification and Reliability*, 15 (2): 97–133, 2005.

[40] J. Mayer. On testing image processing applications with statistical methods. In *Software Engineering 2005* (*SE 2005*), Lecture Notes in Informatics, pages 69–78. Gesellschaft fu"r Informatik, Bonn, 2005.

[41] J. Mayer and R. Guderlei. An empirical study on the selection of good metamorphic relations. In *Proceedings of the 30th Annual International Computer Software and Applications Conference* (*COMPSAC 2006*), volume 1, pages 475–484. IEEE Computer Society, Los Alamitos, CA, 2006.

[42] S. Melax. A simple, fast, and effective polygon reduction algorithm. *Game Developer Magazine*, pages 44–49, November 1998.

[43] A. Memon, I. Banerjee, and A. Nagarajan. What test oracle should I use for effective GUI testing?. In *Proceedings of the 18th IEEE International Conference on Automated Software Engineering* (*ASE 2003*), pages 164–173. IEEE Computer Society, Los Alamitos, CA, 2003.

[44] A.M. Memon, M.E. Pollack, and M.L. Soffa. Automated test oracles for GUIs. In *Proceedings of the 8th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (*SIGSOFT 2000/FSE-8*) pages 30–39. ACM, New York, NY, 2000.

[45] B. Meyer. *Eiffel: the Language*. Prentice Hall, New York, NY, 1992.

[46] E.B. Moore, A.V. Poliakov, P. Lincoln, and J.F. Brinkley. MindSeer: a portable and extensible tool for visualization of structural and functional neuroimaging data. *BMC Bioinformatics*, 8: 389, 2007.

[47] ISO/IEC. The MPEG Standards. Moving Picture Experts Group. Available at: http://www.chiariglione.org/mpeg/standards.htm. (Last accessed: June 15, 2008.)

[48] A.J. Offutt, A. Lee, G. Rothermel, R.H. Untch, and C. Zapf. An experimental determination of sufficient mutant operators. *ACM Transactions on Software Engineering and Methodology*, 5 (2):99–118, 1996.

[49] T. Ostrand, A. Anodide, H. Foster, and T. Goradia. A visual test development environment for GUI systems. In *Proceedings of the 1998 ACM SIGSOFT International Symposium on Software Testing and Analysis* (*ISSTA 1998*), pages 82–92. *ACM SIGSOFT Software Engineering Notes*, 23 (2): 82–92, 1998.

[50] D.K. Peters and D.L. Parnas. Using test oracles generated from program documentation. *IEEE Transactions on Software Engineering* 24 (3): 161-173, 1998.

[51] A. Podgurski, D. Leon, P. Francis, W. Masri, M. Minch, J. Sun, and B. Wang. Automated support for classifying software failure reports. In *Proceedings of the 25th International Conference on Software Engineering* (*ICSE 2003*), pages 465–475. IEEE Computer Society, Los Alamitos, CA, 2003.

[52] G. Rothermel, R.H. Untch, C. Chu, and M.J. Harrold. Test case prioritization. *IEEE Transactions on Software Engineering*, 27 (10): 929–948, 2001.

[53] S. Rusinkiewicz and M. Levoy. Efficient Variants of the ICP Algorithm. In *Proceedings of the third International Conference on 3D Digital Imaging and Modeling (3DIM 2001)*, pages 145–152, IEEE Computer Society, 2001.

[54] M. Segal and K. Akeley. *The OpenGL Graphics System: a Specification*. Version 2.0. Silicon Graphics, Mountain View, CA, 2004.

[55] G.G. Shulmeyer and T.J. McCabe. The Pareto principle applied to software quality assurance. In *Handbook of Software Quality Assurance*, 3rd Edition, pages 291–328. Prentice Hall, Upper Saddle River, NJ, 1998.

[56] Stanford University. The Stanford 3D Scanning Repository. Available at: http://graphics.stanford.edu/data/3Dscanrep/. (Last accessed: June 27, 2008.)

[57] Y. Sun and E.L. Jones. Specification-driven automated testing of GUI-based Java programs. In *Proceedings of the 42nd Annual Southeast Regional Conference* (*ACM-SE 42*), pages 140–145. ACM, New York, NY, 2004.

[58] M. Vanmali, M. Last, and A. Kandel. Using a neural network in the software testing process. *International Journal of Intelligent Systems*, 17 (1): 45–62, 2002.

[59] Web3D Consortium. X3D International Specification Standards. Available at: http://www.web3d.org/x3d/. (Last accessed: June 27, 2008.)

[60] E.J. Weyuker. On testing non-testable programs. *The Computer Journal*, 25 (4): 465–470, 1982.

[61] Wikipedia. Binary Classification. Available at http://en.wikipedia.org/wiki/Binary_classification. (Last accessed: June 30, 2008.)

[62] Wikipedia. Specificity Tests. Available at http://en.wikipedia.org/wiki/Specificity_%28tests%29. (Last accessed: June 30, 2008.)

[63] S.E. Yoon, C. Lauterbach, and D. Monocha. Rlods: fast lod-based ray tracing of massive models. *The Visual Computer*, 22 (9): 772–784, 2006.

[64] Y. Zhu. Uniform remeshing with an adaptive domain: a new scheme for view-dependent level-of-detail rendering of meshes. *IEEE Transactions on Visualization and Computer Graphics*, 11 (3): 306–316, 2005.