To appear in *Journal of Systems and Software*

Automatic Goal-Oriented Classification of Failure Behaviors for Testing XML-Based Multimedia Software Applications: an Experimental Case Study

W. K. Chan [a], M. Y. Cheng [a], S. C. Cheung [b], and T. H. Tse [a]

[a] *The University of Hong Kong, Pokfulam, Hong Kong*

[b] *Hong Kong University of Science and Technology, Clear Water Bay, Hong Kong*

Abstract

When testing multimedia software applications, we need to overcome important issues such as the forbidding size of the input domains, great difficulties in repeating non-deterministic test outcomes, and the test oracle problem. A statistical testing methodology is proposed. It applies pattern classification techniques enhanced with the notion of test dimensions. Test dimensions are orthogonal properties of associated test cases. Temporal properties are being studied in the experimentation in this paper. For each test dimension, a pattern classifier is trained on the normal and abnormal behaviors. A type of failure is said to be classified if it is recognized by the classifier. Test cases can then be analyzed by the failure pattern recognizers. Experiments show that some test dimensions are more effective than others in failure identification and classification.

Key words: Software testing, test dimensions, multimedia application testing, failure identification, failure classification.

[2] A preliminary version of this paper was presented at the 4th International Conference on Quality Software (QSIC 2004).

[3] This research is supported in part by grants of the Research Grants Council of Hong Kong and The University of Hong Kong.

[4] All correspondence should be address to Dr. T. H. Tse, Department of Computer Science, The University of Hong Kong, Pokfulam, Hong Kong. Email: thtse@cs.hku.hk .

1 Introduction

Multimedia software applications are software applications "that support the integrated processing of several media types with at least one time-dependent medium" (Blakowski and Steinmetz, 1996). They are useful in many important areas such as business and education. For instance, supported by an e-learning environment, students can learn at their own pace. Nevertheless, if various animated instructions and corresponding acoustic narrative comments are not properly organized, learners may be trained incorrectly and the training may be counter-productive. Hence, it is important to ensure the quality of such software. As software testing is recognized as the most viable and the most commonly used approach, we shall restrict our attention to software testing in this paper. Software testing is non-trivial for multimedia software applications. The major difficulties, among others, are as follows:

(i) The size of its input domain may be forbiddingly huge. There is an increasing number of software applications, such as RealPlayer (Bulterman, 2004), that support open standards, such as SMIL (1998; 2005), as parts of their input data profiles. The size of each input data profile is limited only by the local resources available. Also, each placeholder of a media object in an input data profile can be mapped to a number of media object instances with different behaviors and supportive technologies. Furthermore, there are many time-points even within a small time interval. All of these add up to explode the size of the input domain.

(ii) It is usually difficult to repeat test cases of such applications to produce identical outcomes. Largely speaking, fuzziness in multimedia software applications is caused by uncertainties or uncontrollable factors in their environments. They are known as common cause variations (Russell et al., 2000). On the other hand, because of reasons such as the huge setup cost of high-precision equipment, we cannot expect all environmental settings to be recorded precisely and reproduced exactly for the testing of multimedia applications. In most cases, testers can only turn a blind eye to such non-deterministic time-dependent fluctuations.

(iii) Because of the difficulties in recording and reproducing environmental settings during executions of a test case, testers may only have a blurred picture of a particular test case. It poses difficulties in verifying test outcomes. In general, this kind of problem is referred to as the *test oracle problem* (Chen et al., 2003).

A testing methodology is proposed and described in this paper. It applies pattern classification techniques and takes test dimensions such as temporal relationships into account as the two major tools to alleviate the problems in testing XML-based multimedia software applications. Its contributions include:

(a) A Step Toward Effective Failure Pattern Recognition: Cunningham and MacKinnon (1998) categorize visual defect metrology into three levels. We shall

adapt their categorization, which was originally used for the recognition of defects in semiconductors, to the recognition of software failures in the following way:

The basic level is *point*. It ignores both failure clustering and special patterns of failures. This is similar to the conventional approaches (Cheung et al., 2004; Lu, 1996) to be described in Section 2.2. They use standard statistical rules to classify failures. The next level is *cluster*. It considers failure clustering, but not special patterns of failures. General-purpose pattern classifiers such as those to be described in Section 2.3 are classified into this category. The ultimate level is *pattern*. Chen and Liu (2000) suggested that it is not a trivial task to classify defect spatial patterns into specific patterns.

In general, the pattern level is more effective than the cluster level, which, in turn, is more effective than the point level. Moreover, a failure pattern signature may appear in situations that satisfy particular properties, but not others. Our present work captures properties as *test dimensions*, and chooses temporal properties as the first kind of dimension to study. It is a necessary step toward the formulation of effective special-purpose pattern recognition algorithms to detect failure patterns for multimedia software applications.

(b) Pattern Classifiers as Statistical Test Oracles: A conventional test oracle (Beizer, 1990) for a software application is used to determine whether a test result is correct. On the other hand, a statistical test oracle, such as those described in Hoffman (1999), may occasionally announce a false result. It is, nevertheless, an alternative if a conventional test oracle is not available or too costly to be used. Since statistical test oracle is not totally reliable, a key research challenge is to improve its accuracy. We propose to train pattern classifiers to serve as statistical test oracles. Given suitable training data sets, the pattern recognition approach is better than the random approach in the accuracy for identifying and classifying failures.

We would like to add that our methodology deals only with selected test dimensions. When a test case with an *unknown* type of pattern is given to a pattern classifier (that is, when there is no "familiar" data available for training), it is not known whether we expect it to be a successful test case or a failure-causing input. As a result, we cannot tell whether any identified failure is legitimate. This is an open and classic oracle problem in software testing and is beyond the scope of the present paper.

(c) Testing by Test Dimensions to Improve Classification Accuracy: To organize test cases into groups (that is, test dimensions), our method uses the fundamental set of generic properties to distinguish temporal relationships in the experimentation. In general, a test case may have multiple test dimensions. Our results, presented in this paper, outperform heuristic identification of failures followed by random classification to a failure class. In terms of effectiveness, it is comparable to the results when all test dimensions are learned by a classifier at the same time. It shows that some test dimensions are more effective in classifying failures.

The rest of the paper is organized as follows: Section 2 provides readers with background information related to our work. Section 3 proposes a methodology for

distinguishing normal and failure behaviors, and classifying the latter. A case study of the methodology and its results are described in Sections 4 and 5, respectively. Finally, Section 6 concludes the paper.

2 Background Information

2.1 Temporal Relationships

Temporal relationships are important attributes in multimedia software applications (Blakowski and Steinmetz, 1996; Cheung et al., 2004). Allen (1983) proposed seven types of temporal relationships between two media objects. Cheung et al. (2004) proposed a more fundamental set of *generic temporal relationships* that can simulate the relations in Allen (1983). In this paper, we shall restrict our scope to the three generic temporal relationships proposed in Cheung et al. (2004). Figure 1 shows a diagrammatic representation of the three generic temporal relationships, which include:

A ***before*** *B***:** Media object *A* terminating before media object *B* begins;
A ***starts_after*** *B***:** Media object *A* starting after media object *B* has begun; and
A ***ends_ahead_of*** *B***:** Media object *B* terminating before media object *A* has ended.

There are other more formal ways to express temporal relations, such as temporal logic, and communicating finite state machines. For example, Little and Ghafoor (1993) have proposed a timed Petri-net model to describe the synchronizations in multimedia data in an application. They focus, however, on the storage and retrieval of media objects in databases.

2.2 Testing Multimedia Software Applications

A conventional testing technique for multimedia software applications is to set an acceptable range of values. Lu et al. (1996) recommended to use 80 ms for audio and video synchronizations. It is subjective to set a suitable zone, however. Cheung et al. (2004) proposed to use statistical approaches to evaluate relevant quantities such as the difference between the start times of two media objects, and expressed the results in terms of confidence levels. Both Cheung et al. (2004) and Lu et al. (1996) are *point*-level failure recognition techniques. Campos et al. (1999) performed formal proving of temporal properties. Based on insights from

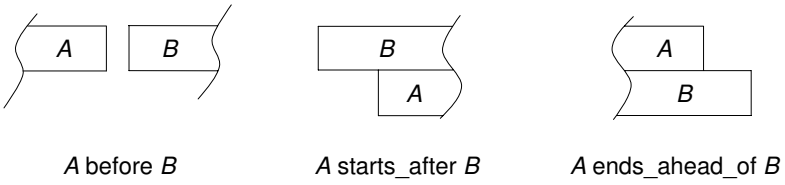

Fig. 1. The three major temporal relationships

the modeling, they reverted to expert judgment mechanisms to improve on the design of a video server component. Zhang and Cheung (2002) used Petri nets to specify temporal relationships and tested multimedia programs against them. There has also been research to generate test cases for conformance and interoperability testing (Hao et al., 2004).

Our work does not depend on any program structures or specifications, and also makes no assumption on other components in the program environment. Thus, the research efforts by the above authors are complimentary to our proposed technique.

2.3 Pattern Classification Techniques

In this section, three representative and popular techniques in pattern classification (Duda et al., 2000) will be described. They serve as the foundation of our present work.

***k*-Nearest Neighbor (KNN):** *k-nearest neighbor* is a non-parametric classification technique, which does not make any assumption about the underlying distribution of the data (Duda et al., 2000). Given n training data sets, let $\boldsymbol{x_i} = (a_i^{(1)}, a_i^{(2)}, \ldots, a_i^{(m)})$ represent the values of the attributes of the ith data set, where $i = 1, 2, \ldots, n$. A query with respect to a test case $\boldsymbol{x} = (a^{(1)}, a^{(2)}, \ldots, a^{(m)})$ can be input to the classifier, which will find a set $S' = \{\boldsymbol{x'_1}, \boldsymbol{x'_2}, \ldots, \boldsymbol{x'_k}\}$ of k training data from the training set $S = \{\boldsymbol{x_1}, \boldsymbol{x_2}, \ldots, \boldsymbol{x_n}\}$ such that $\max_{\boldsymbol{x'_i} \in S'} d(\boldsymbol{x'_i}, \boldsymbol{x}) \leq \min_{\boldsymbol{x_i} \in S \setminus S'} d(\boldsymbol{x_i}, \boldsymbol{x})$, where $d(\,)$ is the distance function.

In our experiments, the KNN classifier is implemented in C++ by one of the authors (Cheng) according to the above description. It uses standard Euclidean distance as the distance function. We have chosen k to be 3 after a careful feasibility trial.

Bayesian Networks: A Bayesian network (Duda et al., 2000) for a set of variables $U = \{u_1, u_2, \ldots, u_n\}$ is a tuple (G, Θ). $G = (V, E)$ is a directed acyclic graph, where (i) there is a one-to-one-correspondence between the elements in V and those in U, and (ii) E $(\subseteq V \times V)$ is a set of directed edges. Θ is a set of conditional probability distributions such that each Θ_i in Θ defines the conditional probability of node u_i given its parents in G. A Bayesian network B represents a joint distribution over U in G as follows: $probability_B(u_1, u_2, \ldots, u_n) = \prod_{i=1}^{n} probability(u_i | \pi_{u_i}, \Theta_i)$, where π_{u_i} is the set of parents of node v_i in V. After a Bayesian network has learned its set of conditional probabilities Θ based on the inter-relationships among given attributes in the training data set U, it can be used to perform pattern classification for new data. In our experiments, Belief Network PowerPredictor (2001) is used.

Neural Networks: *Neural networks* (Mitchell, 1997) are designed to model the human brain. A *perceptron* sums up a number of weighted inputs $(x_i)_{i=1,2,\ldots,n}$ to give $net = \Sigma_{i=0}^{n} w_i x_i$, and generates an output y via a transfer function. Initially,

the network chooses a set of random weights for each node. During the training phase, inputs are fed into the network and the actual outputs are compared with the expected results to estimate the errors. The network then adjusts the weights according to the feedback from the estimated errors. In our experiment, NeuroSolutions (2004) is used.

2.4 Detection of Failures without a Precise Specification

Techniques for detecting software failures without a precise specification fall into two categories: analytical and statistical. Some examples of the former type include program checkers (Blum and Kannan, 1995), and metamorphic testing (Chen et al., 2003).

In this paper, we are more interested in statistical testing and, hence, we shall outline a couple of statistical techniques. Podgurski et al. (2003) applied data clustering analysis to recognize failure behaviors except identify failures. Their subject is a deterministic program. Similarly, Bowring et al. (2004) represented the execution history of each test case of a deterministic program by means of a Markov model. They also used a clustering technique to merge classes into clusters of classes to formulate their pattern classifiers. Both are *cluster*-level failure recognition techniques.

We proposed in Cheng et al. (2004) to let classifiers learn the *patterns* from various categories of normal and faulty multimedia software applications. In this extended version, besides reporting the original findings, we further present the concept of test dimensions, with a long-term objective of enhancing the methodology to achieve *pattern*-level failure recognition.

3 The Testing Methodology

In the present and the next sections, we introduce our testing methodology and then describe an experimental study on the methodology. The major steps of the testing methodology are as follows:

(1) Determine a set Θ of application attributes for pattern classification such that Θ includes the implementations of the three generic temporal relations.
(2) Define the criterion of each test dimension and ensure that the test dimensions collectively cover all generic temporal relations.
(3) Based on the temporal relations on media objects defined in the SMIL-based input data profile in each test case, assign every training test case to at least one test dimension according to the criteria. Hence, form a data set Φ for each test dimension.
(4) Prepare the classes Γ for pattern classification. Each class in Γ is expected to represent a normal case or a kind of failure, but not both.

(5) Based on the attributes set Θ, for each data set Φ, train a pattern classifier to distinguish classes in Γ.
(6) Generate a test data set.
(7) Classify test data set so that each test case is labeled with a class in Γ.
(8) Report the class of any test case if the class represents a kind of failure.
(9) Apply *pattern*-level failure recognizers to find the patterns from a class of failure test cases.

The purpose of Step (1) is to find several appropriate attributes for pattern classification. In Experiment 2, we describe our experience in dealing with attributes in multimedia software applications. Training data are further organized so that they are orthogonal in temporal relations. We envisage that a detectable temporal behavioral failure will be in its most detectable form if we can tune the reference axes into orthogonal forms that cover all the temporal relations. The set of the three generic temporal relations is one of the possible choices. Also, although we have chosen three data sets in Experiments 1 and 2, the value "3" is not absolute. Testers can choose their own numbers of data sets. Step (9) is a longer-term objective that is not covered in this paper. The rest follows a standard pattern classification procedure. We refer readers to (Duda et al., 2000) for more details.

4 The Case Study

In this section, the methodology is investigated by an experimental case study. Three sets of experiments are presented in this paper.

4.1 The Subject X-Smiles and its Testing Environment

As multimedia software applications can be standalone or distributed, we choose *X-Smiles* (2002) version 0.71, an open-source Java-based XML-browser, as the subject of the case study. In this way, although the case study is conducted in offline mode [5], it provides opportunities for extending the work to the testing of distributed multimedia applications.

The Java-based XML-browser, the subject of our study, consists of 55 117 lines of code, 1 010 classes, 168 interfaces, and 5 022 methods [6]. Furthermore, there are many possible behaviors that a software system may exhibit correctly or incorrectly. To reduce the scope of the initial study, we would like to pick the java classes that are most likely to influence temporal constraints. The classes Scheduler.java, BrowserLogic.java, and ElementBasicTimeImpl.java have been selected after a careful evaluation. They contain 152, 198, and 1 032 lines of code, respectively. The sizes of these fragments of code are comparable to the well-known Siemens

[5] In the sense that it is treated as a standalone program running on a machine.
[6] The statistics are collected by the tool LOCC release 4.3.7 available at http://csdl.ics.hawaii.edu/Tools/LOCC.

Experiment	Training Phase	Testing Phase
1	Machine *A*	Machine *A*
2	N/A	Machine *A*
3	Machine *A*	Machine *B*

Machine *A*: Pentium Celeron 500MHz processor, 256 MB of memory, Creative SB Live! Wave Device, and Microsoft Windows 2003 Enterprise Edition.

Machine *B*: Pentium Celeron 550MHz processor, 512 MB of memory, Creative SB Live! Wave Device, and Microsoft Windows 2003 Enterprise Edition.

Table 1
Machine configurations in experiments

Test Suite, a collection of subject programs used in software testing researches in the past 14 years. Both (a) Java Development Kit 1.3 and (b) Java Media Framework 2.1.1 are installed. Table 1 shows the configurations of the machines used for all the experiments in the case study.

4.2 Test Cases and Test Dimensions for X-Smiles

Each test case to be executed by the *X-Smiles* system consists of three components: a SMIL-based (SMIL, 1998) input data profile, a slide show, and an audio narration. The slide show is an animated GIF with a size of 640 $\times$ 480 pixels, and the audio narration is in WAV format with a sampling rate of 44.1 kHz. A sample illustrating a parallel execution of two media objects is shown in Figure 2. It specifies that the animation "*description*" starts its presentation 1 second after the end of the audio object "*media*1".

The test dimensions for the case study are as follows:

***TD* 1 (Simultaneous):** Slide show *starts_after* audio narration by 0 delay and slide show *ends_ahead_of* audio narration also by 0 delay;

***TD* 2 (One During Another):** Audio narration *starts_after* slide show and audio narration *ends_ahead_of* slide show, and the delays in either "*starts_after*" or "*ends_ahead_of*" cannot be both 0 at the same time; and

***TD* 3 (One Before Another):** Slide show *before* audio narration.

4.3 Attributes for the Failure Classification Experiments

In a training phase, data will be collected from a reference system, and from different classes of faulty systems. We have verified the data manually before using them for training. For each system, attribute values are collected in 20 repeated runs per input. The output of each repeated run is considered as a distinct sample data point for pattern classifiers so that the latter can learn the *fuzziness* in behaviors when testing the *X-Smiles* system. Five raw attributes are collected for every training

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<smil>
<head>
<layout>
     <root-layout id="testing" height="600" width="200"
                  background-color="#ffffff" />
     <region id="region1" height="250" width="250" top="50" left="50" />
     <region id="region2" height="250" width="250" top="50" left="300" />
     </layout>
     </head>
<body>
<par id="par1" begin="0.0s" dur="20.0s">
     <audio id="media1" region="region1" src="sound.wav"
            begin="1.0s" dur="5.0s" />
     <img id="description" region="region2" src="description.gif"
          begin="media1.end+1s" dur="5.0s" />
     </par>
     </body>
     </smil>
```

Fig. 2. Sample SMIL-based input data profile

sample data and test case for pattern classification. They are selected from 24 candidate "shopping list" (Meyer, 1997) attributes. In the formulation of the shopping list, we have exercised measures to refer to published work in multimedia system benchmarking and picked the relevant attributes accordingly.

For *TD*1 and *TD*2, the selected attributes include the duration of the slide show, the duration of the audio narration, the difference in start times between the slide show and its audio narration, and the difference in end times between the slide show and its audio narration. For *TD*3, the attributes include the duration of the slide show, the duration of the audio narration, the start time of the slide show, and the difference between the end time of the slide show and the start time of the audio narration. The maximum memory utilization of the system is also recorded in all cases.

4.4 Systems to Generate Behaviors for Pattern Classifications

In this section, we describe the systems that represent normal behaviors and known types of error [7] . We use more than one kind of error so that our investigations will not be biased toward errors of a particular kind. All types of error are, intuitively, related to temporal properties. However, we have exercised measures to ensure that the relationships among types of error are unknown to both test case selection and

[7] For unknown types of error, we refer readers back to the discussions of pattern classifiers as statistical test oracles in Section 1 for more details.

formulation of test dimensions. In fact, it remains an open problem to find the actual relationships among these types of error.

The performances of the reference system used for the training include both the situations when the CPU is almost idle and when it is extremely busy. To simulate a low CPU utilization situation, all other utilities and applications are turned off. To simulate a high CPU utilization situation, a CPU-hungry utility is kept running to ensure that the system is heavily loaded. Three different types of error have been seeded into the reference system. Together with the reference system, there are four kinds of system in total:

***Class* 0:** This is the reference system. The outcomes of the training test cases are checked manually.

***Class* 1:** A priority error is simulated. It reduces the priority of execution threads. It simulates an inappropriate priority configuration in multimedia software applications. Errors are seeded in the file Scheduler.java.

***Class* 2:** A pagination error is simulated. It causes *X-Smiles* to consume a large amount of system memory and, hence, the system will have excessive pagination. Errors are seeded in the file BrowserLogic.java such that it calls another java class MemoryConsumer to simulate the desired failures.

***Class* 3:** A start time error is simulated. It delays the start time of one given media object by a fixed parameter (relative to the SMIL-based input data profile). Errors are seeded in the file ElementBasicTimeImpl.java.

4.5 Experiment 1: Effects of Testing by Test Dimensions

The main objective of this experiment is to find out whether a specific test dimension can classify a failure as effectively as the situation when all test dimensions are used. Intuitively, in the absence of full information, the failure classification rate of the former should be lower than the latter. The question is, to what extent. Part of this experiment was reported in the preliminary version (Cheng et al., 2004) of the present paper.

Experiment 1 is divided into three parts: a main experiment E1 and two reference experiments R1A and R1B.

Experiment E1 (Testing by Test Dimensions): 480 data have been used for training and 1 760 data for testing. For the training phase, 20 repeated experiments have been carried out for each combination of high-low CPU utilizations, the 3 sets of SMIL specifications (that is, test dimensions), and the 4 types of systems. We have re-run the same experiments 20 times to allow for common cause variations. The same training set is used in all the classification techniques to ensure a fair comparison. In the testing phase, we have grouped multiple runs of the same experiment as a data suite.

We also conduct two reference experiments for the purpose of comparison.

Reference Experiment R1A (Conventional Pattern Classification): The first control experiment shows the situation where all the data for every test dimension are grouped together homogeneously and, hence, represents a conventional pattern classification approach. We group all the training test cases into one data set, and all the testing test cases into another data set. For each pattern classifier, we apply the entire training data set in the training phase, and let the classifiers process the entire testing data set.

Reference Experiment R1B (Conventional Approach to Test Multimedia Systems): The second control experiment shows the situation where conventional techniques are used (see Section 2.2). We use the suggestions of Lu et al. (1996) to determine whether a failure is identified in a test outcome. We note that this is not a pattern classification experiment.

4.6 Experiment 2: Natural Variations of Data Attributes

One of the major assumptions of our method is that uncertainty variations in behaviors constitute part of the output of a test case. They are generally found to be difficult to model and study analytically. Hence, this paper opts for a statistical approach. Thus, a case study will be irrelevant to our method if there happens to be very few behavioral uncertainties.

Experiment 2 consists only of one part: a main experiment E2.

Experiment E2 (Fuzziness of Attributes for Multimedia Software Applications): It is designed to examine the behavioral uncertainties of X-Smiles. It repeatedly applies a SMIL test case (see Section 4.2) consisting of two media objects to system *Class*0 (see Section 4.4) for 240 times in a low CPU utilization environment. The same experiment is repeated in a high CPU utilization environment. Hence, 480 sets of attributes are collected to produce the results in Section 4.6. The SMIL file defines that the two objects (denoted as Object 1 and Object 2 in this paper) are in the same concurrent execution block. The two objects are specified to start and end simultaneously. The duration of each media object is 30 seconds. Four attributes which are tightly coupled to generic temporal relations (see Section 4.3) are collected for the experiments.

4.7 Experiment 3: Effects of Classifications Based on Almost-Identical Hardware Configurations

The testing of the same software may, in practice, be conducted on more than one machine. Since a typical multimedia software application may exhibit non-deterministic behaviors, a strategy to reduce fluctuations across different hardware configurations is to use a number of almost-identical [8] machines, such as using 500 units of Dell Precision™ 670 to conduct testing at the same time. Moreover,

[8] There are intrinsic variations in hardware operations. Hence, each machine is unique from this perspective.

Classification Method	*TD* 1	*TD* 2	*TD* 3	Mean
Bayesian Networks	71.4%	75.0%	96.4%	81.0%
***k*-Nearest Neighbor**	78.6%	84.4%	92.9%	85.3%
Neural Networks	75.0%	78.1%	92.8%	82.0%
Mean	75.0%	79.2%	94.0%	82.7%

Table 2
Successful test results of Experiment E1 under high CPU utilization, by test dimensions with data normalization

testing is conducted in many rounds in a typical iterative approach to software development. Machines available for an earlier round may not be available for a subsequent round. It would be a pity if a tester could not use a previous test suite to regress the software. A main objective of this experiment is to start the initial work to investigate whether failure identification and classification across machines can be effective.

Experiment 3 is divided into two parts: a reference experiment R3 and a main experiment E3.

Reference Experiment R3 (Effects of Classifications on the Same Machine): Since this experiment is basically repeating that presented in Cheng et al. (2004), we have reduced the scale to only three SMIL files in each round [9]. It uses a SMIL file from each of *TD*1, *TD*2, and *TD*3 as inputs to the systems *Class*0, *Class*1, *Class*2, and *Class*3 to produce the five attributes for classification under a low CPU utilization environment. For each test input, we collect the corresponding output attribute values 20 times for both the training and testing phases. The same experiment is repeated in a high CPU utilization environment.

Experiment E3 (Effects of Classifications on an Almost-Identical Machine): It repeats Experiment R3, but uses an *almost-identical* machine in its testing phase. Details of the machines can be found in Table 1.

5 Results of the Case Study

5.1 Results of Experiment 1: Effects of Testing by Test Dimensions

Tables 2 and 3 summarize the results of Experiment E1 under high and low CPU utilizations, respectively, with data normalization. They show the percentages of test cases that are correctly classified under each test dimension.

[9] Nevertheless, it still fits the aim of the experimental study. This is because if the results of behavioral classification in Experiments R3 and E3 were unsatisfactory for identical inputs, it would be pointless to conduct follow-up experiments on a larger scale.

Classification Method	*TD* 1	*TD* 2	*TD* 3	Mean
Bayesian Networks	64.3%	75.0%	60.7%	66.7%
***k*-Nearest Neighbor**	64.3%	75.0%	64.3%	67.9%
Neural Networks	71.4%	62.5%	60.7%	64.9%
Mean	66.7%	70.8%	61.9%	66.5%

Table 3
Successful test results of Experiment E1 under low CPU utilization, by test dimensions with data normalization

(*a*) In general, the mean results of the three classification techniques exhibit more or less the same percentage of accuracy in failure identification. We find that it is more likely to identify failures under high CPU utilization than under low utilization.

To confirm the results, we have re-developed test sets conforming to SMIL 2.0 standards (2005), and using Allen's set of temporal relations (Allen, 1983) separately. Then, we re-run the part of Experiment E1 which is under low CPU utilization. The results agree with the results in Table 3 with only marginal differences. There are other rooms of improvements to improve classification rates, which have been discussed in the preliminary version of this paper (Cheng et al., 2004) and are not repeated here.

(*b*) However, the mean results of the three test dimensions show significant deviations. In particular, *TD*3 under high CPU utilization outperforms the average in all cases, and is remarkably better than its counterpart under low CPU utilization. Furthermore, *TD*1 requires 0 second delay in either *start_after* or *ends_ahead_of* relation in a test case. It is more stringent than the requirements of $TD2$ and $TD3$. Intuitively, any failure of a more stringent requirement should be easier to reveal. On the contrary, the experimental results of the test dimension $TD1$ are generally poorer than the others. In other words, intuitively intricate test cases may not show advantages in detecting failures

(*c*) The mean results of Experiments R1A (which does not use the concept of test dimensions in classifying failures) and R1B (which uses an ad hoc method to identify failures) are 69.9% and 54.7%, respectively, under low CPU utilization. Compared with Table 3, they show that the test dimension method only marginally reduces the resolution of failure classification ($69.9\% - 66.5\% = 3.4\%$), but significantly outperforms the ad hoc failure identification approach ($66.5\% - 54.7\% = 11.8\%$).

5.2 Results of Experiment 2: Natural Variations of Data Attributes

Figure 3 shows the variations of the data attributes. The left column shows the set of attributes, including the time difference in "Object 2 *starts_after* Object 1" (L1),

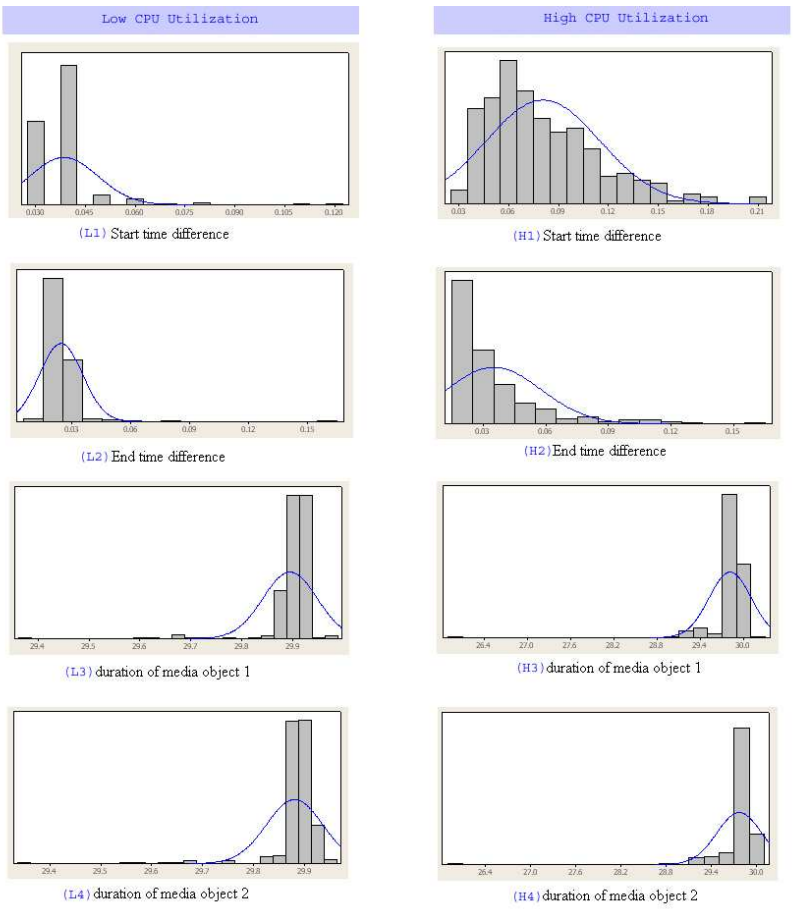

Fig. 3. Natural variations of attributes in the case study

the time difference in "Object 2 *ends_ahead_of* Object 1" (L2), the presentation duration of Object 1 (L3), and the presentation duration of Object 2 (L4), when the experiment is conducted in a low CPU utilization environment. The right column shows the same set of attributes ((H1) to (H4)) in a high CPU utilization environment. We also observe the following trends from Figure 3:

(a) Not every *obvious candidate* of time-dependent attributes demonstrates a wider spread of variations under high CPU utilization than under low CPU utilization. The variations of presentation durations in cases (L3) and (L4) are larger than those in cases (H3) and (H4). This observation is interesting and worth further investigation. Furthermore, the temporal relations *starts_after* and *ends_ahead_of* ((L1) versus (H1) and (L2) versus (H2)) follow our intuition that wider spreads in variations of values are recorded under a tighter resource environment. Choosing *starts_after* and *ends_ahead_of* temporal relationships for further study may enable researchers to have a more theoretical understanding of the so-called *common cause variations*.

(b) All the diagrams look like unipolar distributions. In particular, cases (H1) and (H2) look like Gaussian and Poisson distributions, respectively. They agree

Classification Method	*TD* 1	*TD* 2	*TD* 3	Mean
Bayesian Networks	100.0%	75.0%	100.0%	91.7%
***k*-Nearest Neighbor**	100.0%	75.0%	100.0%	91.7%
Neural Networks	100.0%	87.5%	100.0%	95.8%

Table 4
Successful test results of Experiment R3 under high CPU utilization, by classes of test dimensions

Classification Method	*TD* 1	*TD* 2	*TD* 3	Mean
Bayesian Networks	62.5%	75.0%	100.0%	79.2%
***k*-Nearest Neighbor**	75.0%	100.0%	100.0%	91.7%
Neural Networks	56.3%	87.5%	100.0%	81.3%

Table 5
Successful test results of Experiment E3 under high CPU utilization, by classes of test dimensions

Classification Method	*TD* 1	*TD* 2	*TD* 3	Mean of E3	Mean of R3
Bayesian Networks	68.8%	87.5%	68.8%	75.0%	70.8%
***k*-Nearest Neighbor**	62.5%	81.3%	75.0%	72.9%	70.8%
Neural Networks	56.3%	75.0%	68.8%	66.7%	66.7%

Table 6
Successful test results of Experiments E3 and R3 under low CPU utilization, by classes of test dimensions

with Cheung et al. (2004) that some common cause variations can be described using statistical distribution models. It also helps to explain why Experiment R1B using the *point*-level failure recognition approach can identify failures in more than a half of all cases.

5.3 Results of Experiment 3: Effects on Classifications Based on Almost-Identical Hardware Configurations

We make the following three observations on Experiment E3:

(a) The test dimension *TD*1 in Tables 4 and 5 show a drop of about 35% in classification rates. Thus, a test dimension that has a very good failure detection capability on one machine may not be assumed to be effective in classifying test

cases from other machines.

(b) Comparing the changes in the mean results between Experiments R3 and E3 under low CPU utilization in Table 6 (69.4% versus 71.5%), and between Tables 4 and 5 (93.1% versus 79.2%), the low CPU utilization situation provides a less dynamic change in success rates of classifications. It means that if the goal of a failure classification is to compare the failure rates with different tests of the same application on *almost-identical* machines, it would be better to conduct the testing of multimedia software applications in a low CPU utilization environment, because different hardware and software environmental configurations may render the CPU utilization into unforeseeable situations during tests. On the other hand, if the objective of a test is to find failures for software debugging, one can conduct testing in a high CPU utilization situation. Experiment E3 shows a loss of about 10% in classification capability.

(c) Experiment E3 preliminarily shows that it is feasible to test a multimedia software application on different hardware and software environments. More work is required to make this observation more conclusive. We are conducting additional experiments using more kinds of test case, different machine configurations, and under different kinds of CPU loading.

6 Conclusion

We have studied the difficulties in testing multimedia software applications, which include (i) the forbidding sizes of the input domains, (ii) non-reproducible outcomes of a test case in some situations, and (iii) the test oracle problem. A testing methodology is proposed in this paper to alleviate the problems. Although only SMIL-based multimedia software applications are examined in this paper, our methodology can be applied to other kinds of applications with input data profiles.

We have kicked off the work that moves from *cluster*-level failure recognition techniques toward *pattern*-level for testing multimedia software applications. We start with the concept of test dimensions. We choose temporal properties as the first kind of dimensions to study, as temporal behaviors are an important feature of multimedia applications.

In this paper, three generic temporal relations are used to organize a test set into different orthogonal test dimensions. They are presented as data sets in both the training and testing phases of pattern classification in the case study. We believe that an appropriate set of orthogonal measures would allow a higher resolution of failures in testing multimedia software applications.

Our main experimental results include the following: (a) Pattern classifications with the concept of test dimensions outperform others without this notion. (b) Failures of stringent test cases are less effectively classified. (c) Success in the classification of a test dimension at one machine may not be transferable to another almost-identical machine.

To further study generic kinds of failure in multimedia software applications, we are conducting experiments, in a larger scale, on the use of mutation analysis for the classification of failures. There are already quite a lot of promising initial results. The present work is a necessary step for the setting up of a benchmark for comparisons with more complex techniques.

As future work, we shall extend the testing methodology to ease the difficulty in finding classes for pattern classification, to systematize the organization of orthogonal data sets, to improve the resolutions of failures identification, to exploit supports for fault location, and to support iterative approaches of software development and testing. While we have obtained promising initial results in some of these issues, we shall conduct more experiments and report them in the near future.

Acknowledgments

We would like to thank the Program Co-Chairs of the 4th International Conference on Quality Software (QSIC 2004) for inviting us to submit this extended version, and acknowledge the anonymous reviewers for their thorough reviews of the paper.

References

Belief Network (BN) PowerPredictor. 2001. Available at http://www.cs.ualberta.ca/%7Ejcheng/bnpp.htm.

NeuroSolutions. 2004. Available at http://nd.com.

Synchronized Multimedia Integration Language (SMIL) 1.0 Specification. 1998. Available at http://www.w3.org/TR/REC-smil.

Synchronized Multimedia Integration Language (SMIL) 2.0 Specification. 2005. Available at http://www.w3.org/TR/2005/REC-SMIL2-20050107.

X-Smiles. 2002. Available at http://www.xsmiles.org.

Allen, J. F., 1983. Maintaining knowledge about temporal intervals, Communications of the ACM 26, 11, 832–843.

Beizer, B., 1990. Software Testing Techniques, Van Nostrand Reinhold, New York.

Blakowski, G. and Steinmetz, R., 1996. A media synchronization survey: reference model, specification, and case studies, IEEE Journal on Selected Areas in Communications 14, 1, 5–35.

Blum, M. and Kannan, S., 1995. Designing programs that check their work, Journal of the ACM 42, 1, 269–291.

Bowring, J. F., Rehg, J. M., and Harrold, M. J., 2004. Active learning for automatic classification of software behavior. In: Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2004), ACM Press, New York, 195–205.

Bulterman, D. C. A., 2004. A linking and interaction evaluation test set for SMIL. In: Proceedings of the 15th ACM Conference on Hypertext and Hypermedia (HYPERTEXT 2004), ACM Press, New York, 46–47.

Campos, S., Ribeiro-Neto, B., Macedo, A., and Bertini, L., 1999. Formal verification and analysis of multimedia systems. In: Proceedings of the 7th ACM International Conference on Multimedia (Part 1), ACM Press, New York, 419–430.

Chen, F. L. and Liu, S. F., 2000. A neural-network approach to recognize defect spatial pattern in semiconductor fabrication, IEEE Transactions on Semiconductor Manufacturing 13, 3, 366–373.

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

Cheng, M. Y., Cheung, S. C., and Tse, T. H., 2004. Towards the application of classification techniques to test and identify faults in multimedia systems. In: Proceedings of the 4th International Conference on Quality Software (QSIC 2004), IEEE Computer Society Press, Los Alamitos, California, 32–40.

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

Cunningham, S. P. and MacKinnon, S., 1998. Statistical methods for visual defect metrology, IEEE Transactions on Semiconductor Manufacturing 11, 1, 48–53.

Duda, R. O., Hart, P. E., and Stork, D. G., 2000. Pattern Classification, Wiley, New York.

Hao, R., Lee, D., Sinha, R. K., and Griffeth, N., 2004. Integrated system interoperability testing with applications to VoIP, IEEE/ACM Transactions on Networking 12, 5, 823–836.

Hoffman, D., 1999. Heuristic test oracles, Software Testing and Quality Engineering 1, 2, 29–32.

Little, T. D. C. and Ghafoor, A., 1993. Interval-based conceptual models for time-dependent multimedia data, IEEE Transactions on Knowledge and Data Engineering 5, 4, 551–563.

Lu, G., 1996. Communication and Computing for Distributed Multimedia Systems, Artech House, Boston, Massachusetts.

Meyer, B., 1997. Object-Oriented Software Construction. Prentice Hall International Series in Computer Science, Prentice Hall, Hemel Hempstead, Hertfordshire, UK.

Mitchell, T. M., 1997. Machine Learning, McGraw-Hill, New York.

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

Russell, E. L., Chiang, L. H., and Braatz, R. D., 2000. Data-Driven Techniques for Fault Detection and Diagnosis in Chemical Process, Springer, Berlin.

Zhang, J. and Cheung, S. C., 2002. Automated test case generation for the stress testing of multimedia systems, Software: Practice and Experience 32, 15, 1411–1435.