

# Regression Testing of Classes based on TCOZ Specification

Hui Liang  
School of Computing  
National University of Singapore  
lianghui@comp.nus.edu.sg

## Abstract

*Regression testing is an important software maintenance activity. The existing regression testing techniques are mostly code-based. This paper presents a specification-based regression testing technique for classes specified in Timed Communicating Object-Z (TCOZ) notation, which is an integrated formal modelling notation for the design of complex systems. To reduce the cost of regression testing, a TCOZ-based regression test selection algorithm is presented; it selects test cases relevant to the changed specifications from the original test suite for use in regression testing. This paper also describes TcozRts, a TCOZ specification-based regression testing system that implements our technique. The technique is strictly specification-based, and it doesn't require any assumptions about the programming language that is used to implement the specification.*

## 1. Introduction

Regression testing is an important software maintenance activity which provides the confidence that the changed parts of the software system behave as intended. Reusing the test suite that was used to test the original version of the system can reduce the cost and effort required by regression testing. However, rerunning all of the test cases in the original test suite may still take much cost. Moreover, some of the test cases in the original test suite may be obsolete to the modified version of the system and can not be used for regression testing. A solution to this problem is to apply regression test selection techniques to select a proper subset of the test suite for regression testing. Most of the regression test selection techniques that have been developed for testing procedural languages [2, 6, 14, 23, 27, 29] and for testing object-oriented languages [11, 24, 28, 10] are strictly code-based. However, when selecting regression test cases, both specification-based and code-based techniques should be employed because if the specifications for the software

have been changed while code modifications necessary to implement the changed specifications have not been made, such a fault can only be detected by specification-based test case selection which selects the test cases related to the changed specifications [24]. Meanwhile, new test cases might well be required in regression testing because of the changes made to the software systems. With the solid mathematical bases, formal specification provides a good starting point for the systematic generation of new test cases.

In this paper, we present a technique for the regression testing of classes based on Timed Communicating Object-Z (TCOZ) [18, 19] specification. TCOZ is a modelling notation built on Object-Z's [5] strengths in modelling complex data and algorithms, and on Timed CSP's [25] strengths in modelling process control and real-time interactions. It provides a timed, multi-threaded object modelling notation for the design of complex systems. The strengths of and links between TCOZ and Timed Automata are investigated in [7]. The way of generating system test requirements from TCOZ is explored in [8]. [26] presents approaches of using XML/XSL as a transformation tool to visualize TCOZ models into various UML diagrams and to animate TCOZ specifications with a multi-paradigm programming language - Oz.

The regression testing technique presented in this paper addresses the regression test selection problem and test suite augmentation problem [10] for the regression testing of classes specified in TCOZ notation. It uses the information about the modifications made to the specification to identify the obsolete test cases, to select the test cases related to the changed specifications from the original test suite, and to guide the generation of new test cases for regression testing.

The technique is strictly specification-based. No complex static or dynamic code analysis is required. Therefore, it is independent of the programming language that is used to implement the specification. It can be fully automated. A TCOZ-based regression testing system named TcozRts is described in this paper; it implements our specification-based technique for the regression testing of class.

The rest of the paper is organized as follows. In the next section, TCOZ notation is briefly introduced and the characteristics of the class specified in TCOZ notation are discussed. In Section 3, we present a graphic representation for the class which is specified in TCOZ notation. The technique for regression testing of class based on TCOZ specification is presented in Section 4. In Section 5, we describe a TCOZ-based regression testing system in detail. We present the related research in Section 6. Finally, we conclude in Section 7.

## 2. Classes in TCOZ

Timed Communicating Object-Z (TCOZ) [18, 19] is essentially a blending of Object-Z [5] with Timed CSP [25], preserving them as proper sub-languages of the blended notation. The essence of this blending is the identification of Object-Z operation schemas with terminating CSP processes. Thus operation schemas and CSP processes occupy the same syntactic and semantic category, operation schema expressions may appear wherever processes may appear in CSP and CSP process definitions may appear wherever operation definitions may appear in Object-Z. A detailed introduction to TCOZ and its Timed CSP and Object-Z features can be found in [18]. The formal semantics of TCOZ is documented in [17].

The primary specification structuring device in TCOZ is the Object-Z class mechanism. In TCOZ notation, operation schemas (both syntactically and semantically) are identified with (terminating) CSP processes that perform only state update events; active classes are identified with non-terminating CSP processes; the MAIN process determines the behaviour of the objects of an active class after initialization.

An example of the definition of a class in TCOZ specification is shown in Figure 1. It is the specification of a simple timed message queue system. The *Timed Message Queue System* can receive a new message (of type  $[MSG]$ ) through an input channel 'in' or remove a message and send it through an output channel 'out'. If there is no interaction within a certain time ' $T_o$ ', a message will be lost from the current ( $items$ ) list and stored in an asynchronous actuator list ( $lost$ ) so that other objects (un-specified) with a sensor ' $lost$ ' can read it at any time. The messages in the queue are removed in first-in-first-out manner.

## 3. Representation for Class specified in TCOZ

In this section, we give an overview of testchart which is the graphic representation for a class specified in TCOZ notation. We also present the *interaction coverage* criterion; it is proposed based on the testchart and will be used in our regression testing technique.

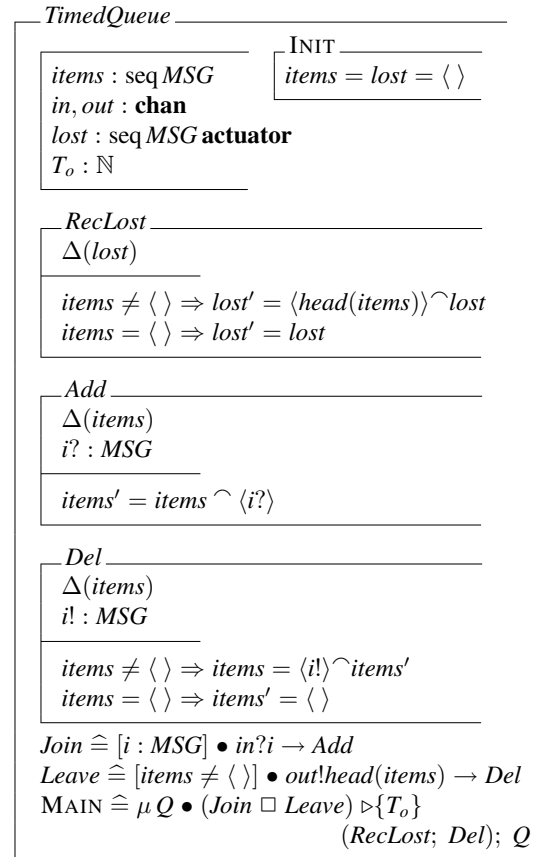


Figure 1. Timed Message Queue System.

### 3.1. Testchart

We developed testchart for TCOZ specification-based class testing where testchart is used for test case generation and coverage evaluation [15]. In this paper, the testchart serves as the base for detecting modifications made to the specification of a class. By representing a class with a testchart, the original and modified versions of the testchart can be compared to reveal the modifications made to the control flow structure of the specification of a class and the information about the modifications will be used to identify the obsolete test cases and to guide the generation of new test cases for regression testing.

The testchart is derived from a class' specification which is written in TCOZ notation. In TCOZ, active classes are identified with non-terminating CSP processes and the MAIN process determines the behaviour of objects of an active class after initialization. This makes it possible to construct a graphic representation for the class, based on the

semantics of TCOZ. Testchart depicts all possible interactions among the operations of a class, just like a control flow graph of program statically represents all possible program paths.

A testchart for a class is a 3-tuple  $\langle S, T, s_0 \rangle$  where

1.  $S$  is a finite, non-empty set of vertices. Each vertex in  $S$  represents an operation of the class. Each of the operations that appear in the definition of process MAIN, MAIN itself included, has a corresponding vertex in the testchart.
2.  $T \subseteq S \times S$  is a set of directed edges between vertices. An edge  $(A, B) \in T$  if and only if it is permissible for a client module to invoke the operation represented by  $A$  followed by operation represented by  $B$ .
3.  $s_0 \in S$  is the vertex that represents the *Init* operation of the class.

The state-guards in TCOZ specification [16] are used to label the corresponding edges in the testchart.

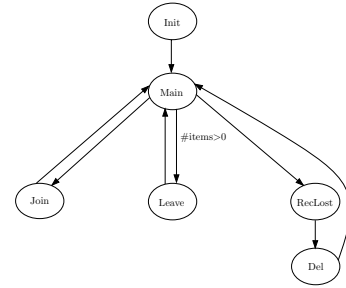
As an example, the testchart for the *Timed Message Queue System* is shown in Figure 2(a). In the TCOZ specification of the *Timed Message Queue System* which is shown in Figure 1, the MAIN process is defined as follows:

$$\text{MAIN} \triangleq \mu Q \bullet (\text{Join} \sqcap \text{Leave}) \triangleright \{T_o\} \\ (\text{RecLost}; \text{Del}); Q$$

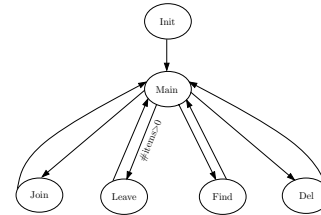
According to the semantics of TCOZ [17], the system will make a choice between operation *Join* and *Leave*, if neither of them is invoked within a certain time ' $T_o$ ' then operation *RecLost* will be invoked followed by operation *Del*. Therefore, in the testchart for the *Timed Message Queue System*, there are directed edges from *Main* to *Join*, *Leave* and *RecLost* respectively and also a directed edge from *RecLost* to *Del*. Since MAIN is a recursively-defined process, there are directed edges from *Join*, *Leave* and *Del* to *Main*. In the definition of operation *Leave*, there is a state guard  $[items \neq \langle \rangle]$ ; so the directed edge from *Main* to *Leave* will be labelled by  $\#items > 0$  which is a variant of  $items \neq \langle \rangle$ . As a result, we get the corresponding testchart as Figure 2(a) shows.

### 3.2. Coverage Criteria

Coverage criteria specify the set of elements to be covered in testing and guide the selection of test cases. For specification-based testing, a few coverage criteria have been proposed. For instance, Offutt and Liu *et al.* [20, 21] defined four criteria at different levels of abstraction on the specifications for the test generation from state-based specifications. Andrews and France *et al.* [1] proposed test adequacy criteria, based on UML model elements, for testing executable forms of UML models.



(a) Original Version



(b) Modified Version

**Figure 2. Testcharts for Timed Message Queue System.**

Class testing typically involves the invocation of sequences of operations (methods) in various orders. In this paper, we concentrate mainly on test cases as sequences of operations and we do not consider input values. The set of test sequences, which satisfies the test requirement for the class-level testing, demands that all the interactions between methods be covered. The directed edges in the testchart correspond to the interactions between the methods specified in TCOZ specifications. Therefore, based on the testchart, we propose *interaction coverage* criterion and we will use it in our regression test technique to guide regression test selection and test cases generation.

**Interaction Coverage** – A set TS of test sequences satisfies the *interaction coverage* criterion if only if for all directed edges  $(a, b) \in T$ , there is at least one test sequence  $t \in TS$  such that  $t$  contains  $(a, b)$ .

### 4. TCOZ-based Regression Testing

In this section, we present a TCOZ specification-based technique for the regression testing of classes. The presentation begins with an overview of the technique. Next, we present the modifications to the specification that we take into consideration in this paper and discuss the impact of those modifications on the original test cases. Then, we illustrate the regression testing technique with an example. Finally, a regression test selection algorithm is presented.

## 4.1. Overview of the Technique

Our regression testing technique functions on testcharts to select the test cases related to the changed specifications from the existing test suite and to generate new test cases for regression testing. It performs the following main steps: (1) it parses the original and modified versions of specifications to identify modified operations, (2) it constructs testcharts to represent the original and modified versions of the class' specifications respectively, (3) it compares the two versions of the testchart to identify the added/deleted operations and the added/deleted interactions, (4) based on the information obtained through steps 1 and 3, it identifies the obsolete test cases and selects the test cases related to the changed specifications, from the original test suite, for use in the regression testing, (5) based on the information obtained through step 3, it generates test cases for testing the new operations and interactions that are added to the specification, so that the interaction coverage criterion will be satisfied in the regression testing.

## 4.2. Modifications to TCOZ Specification

In this paper, we consider the following types of modifications to the specification of a class.

- **Modified operation:** We assume that an operation has a unique name and is not renamed across the different versions of the class' specification unless it doesn't exist in the modified version. In TCOZ, the operations of a class can be defined in the form of operation schema (e.g. *Add* in Figure.1) or in the form of CPS process (e.g. *Join* in Figure.1). The change of an operation may result from the addition/deletion of an attribute or the change of one of the attributes the class can access. In TCOZ specification, the attributes of a class are declared in the state-schema. An added/deleted attribute is an attribute that is not declared in the original/modified version of the specification of a given class, but is declared in the modified/original version of the specification. A modified attribute is an attribute which exists in both versions of the specification but with different scope, type, or visibility. The change of an operation may also result from the change of a precondition or postcondition when it is defined in the form of operation schema, or the change of the process when it is defined in terms of CSP process. The modified operations can be identified automatically by parsing the TCOZ specification of a class.
- **Added/Deleted operation:** An added/deleted operation is an operation that does not exist in the

original/modified version of the class' specification, but exists in the modified/original version of the specification. Note that the function  $V$  returns the set of vertices for the testchart in question; each vertex of a testchart represents an operation of the corresponding class as we have stated in Section 3; *operations-added* is the set of added operations; *operations-deleted* is the set of deleted operations;  $Testchart_o$  and  $Testchart_m$  are the testcharts for the original version and modified version of a class' specification, respectively.

$$\text{operations-added} \leftarrow V(Testchart_m) - V(Testchart_o)$$

$$\text{operations-deleted} \leftarrow V(Testchart_o) - V(Testchart_m)$$

- **Added/Deleted interaction:** An added/deleted interaction is an interaction between the operations of a class, which does not exist in the original/modified version of the class' specification but exists in the modified/original version of the specification. In TCOZ, active classes are identified with non-terminating CSP processes and the MAIN process determines the behaviour of objects of an active class after initialization. The testchart, which is derived from TCOZ specifications of the class under test as we discussed in Section 3, depicts the control flow relationships among the operations of the class. The directed edges in the testchart correspond to the interactions between the operations. Note that the function  $E$  returns the set of edges for the testchart in question; *interactions-added* is the set of added interactions; *interactions-deleted* is the set of deleted interactions;  $Testchart_o$  is the testchart for the original specification of a class and  $Testchart_m$  is the testchart for the modified version of the specification.

$$\text{interactions-added} \leftarrow E(Testchart_m) - E(Testchart_o)$$

$$\text{interactions-deleted} \leftarrow E(Testchart_o) - E(Testchart_m)$$

## 4.3. Impact of Modifications on Test Cases

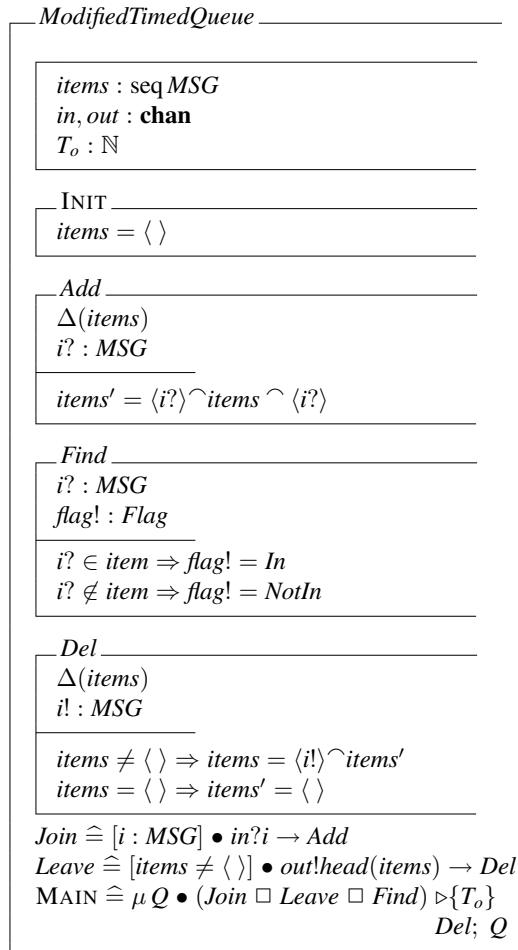
Following the classification presented by Leung and White [13], we classify original test sequences into three categories: obsolete, retestable and reusable.

- **Obsolete:** A test sequence is *obsolete* if it is an invalid sequence of operations in the modified version of the class' specification.
- **Retestable:** A test sequence is *retestable* if it remains valid in the modified version of the class' specification, but one or more of the operations in the sequence have been modified.

- Reusable: A test sequence is *reusable* if it is valid and consists of operations that have remained unchanged in the modified version of the class' specification.

It should be noted that the addition/deletion of operations or interactions will make a test sequence obsolete. It is necessary to identify the obsolete test sequences and remove them from the original test suite if any test sequence reuse is desired. The retestable test sequences must be rerun to ensure that the regression testing is safe while the reusable test sequence do not need to be rerun in regression testing.

$Flag ::= In \mid NotIn$



**Figure 3. Modified Timed Message Queue System.**

#### 4.4. An Example

Figure 1 presents the TCOZ specification of a *Timed Message Queue System*; the modified version of the specification is presented in Figure 3. By parsing the specification, it can be identified that operation *Add* has been modified. In the modified version of the specification, given an  $i$  of type *MSG*, operation *Add* will attach it to both head and tail of the list. *Add* appears in the definition of operation *Join*; the semantics of operation *Join* is modified due to the modification to *Add*. As a result, we obtain the set **operations-modified** = {Add, Join}.

Figure 2(a) and (b) show the original version and modified version of the testchart for the *Timed Message Queue System*, respectively. The original *Timed Message Queue System* has operations: *Join*, *Leave*, *RecLost* and *Del*. The modified version has 3 of the 4 original operations; *RecLost* has been deleted. A new operation *Find* is added. Given an  $i$  of type *MSG*, operation *Find* will find out whether it is in the queue. In the modified version of the *Timed Message Queue System*, the MAIN process is defined as follows:

$$MAIN \hat{=} \mu Q \bullet (Join \square Leave \square Find) \triangleright \{T_o\} \\ Del; Q$$

Correspondingly, in the testchart for the modified *Timed Message Queue System*, the vertex representing operation *RecLost* doesn't exist while a vertex representing operation *Find* is added. There are three new directed edges: (Main, Find), (Find, Main) and (Main, Del). The following four sets can summarize the differences between the two testcharts:

- **operations-added** = {Find}
- **operations-deleted** = {RecLost}
- **interactions-added** = {(Main, Find), (Find, Main), (Main, Del)}
- **interactions-deleted** = {(Main, RecLost), (RecLost, Del)}

$\langle Init; Join; Leave; RecLost; Del \rangle$ ,  $\langle Init; RecLost; Del \rangle$ ,  $\langle Init; Join; Leave \rangle$  are three test sequences from the test suite that is used to test original *Timed Message Queue System*. Since operation *RecLost* has been deleted, operation sequence  $\langle Init; Join; Leave; RecLost; Del \rangle$  becomes invalid for the modified *Timed Message Queue System* and can not be used for regression testing. Because (Main, RecLost) has been deleted, operation sequence  $\langle Init; RecLost; Del \rangle$  becomes obsolete and can not be used for regression testing.  $\langle Init; Join; Leave \rangle$  is selected for regression testing because it is valid for the modified *Timed Message Queue System*.

*Queue System* and one of the operations it includes, *Join*, has been modified. Moreover, new operation *Find* is added; to satisfy the *interaction coverage* criterion, new test sequences such as  $\langle \text{Init}; \text{Join}; \text{Find} \rangle$  need to be generated for regression testing.

#### 4.5. Regression Test Selection Algorithm

We now present a TCOZ specification-based regression test selection algorithm. To identify obsolete test sequences and select test sequences from the original test suite, we associate each test sequence with two bit vectors, *Operations-Used* and *Interactions-Used*. The algorithm is shown in Figure 4. It takes a number of parameters: (1) the original test suite **TS**, (2) the set of modified operations **OpMd**, (3) the set of deleted operations **OpDel**, and (4) the set of deleted interactions **IntrActDel**. It returns **RTS**, the subset of **TS** which is selected for use in regression testing.

The algorithm starts by initializing **Obslt**, which will hold test sequences that are obsolete for regression testing, to *empty*, and initializing **RTS** to *empty*. For each **ts** from **TS**, the algorithm attains the bit vector *Operations-Used* of **ts** (Line 4). If **ts** includes an operation  $O_i$  which does not exist in the new version of the specification (Lines 5, 6), then **ts** is identified as obsolete (Line 7). The algorithm continues by attaining the bit vector *Interactions-Used* of **ts** (Line 10). If **ts** includes an interaction  $\text{IntrAct}_i$  which does not exist in the new version of the specification (Lines 11, 12), then **ts** is identified as obsolete (Line 13). Having identified all the obsolete test sequences, for each **ts** that is an element of **TS** but not an element of **Obslt**, the algorithm attains the bit vector *Operations-Used* of **ts** (Line 18). If **ts** includes an operation  $O_i$  which is modified in the new version of the specification (Lines 19, 20), then **ts** must be selected for regression testing (Line 21).

#### 5. TCOZ-based Regression Testing System

In this section, we present our TCOZ-based regression testing system which is named TcozRts. It is built based on the technique we have presented in the previous sections.

TcozRts takes the original and modified versions of the specification and the original test suite as input, and outputs regression test suite which consists of retestable original test cases and newly generated test cases. TcozRts classifies the original test cases as obsolete, reusable and retestable, based on the information obtained by comparing the original version and modified version of the specifications and testcharts. Moreover, it is likely that new functionalities are added to the system; new test cases that can be used to test those parts of the system should be developed for regression testing; this is referred to as the test suite augmentation

**Algorithm** TestSelection(**TS**, **OpMd**, **OpDel**, **IntrActDel**): **RTS**

**input**

**TS**: original test suite;  
**OpMd**: the set of modified operations;  
**OpDel**: the set of deleted operations;  
**IntrActDel**: the set of deleted interactions

**output**

**RTS**: subset of **TS** selected for use in regression testing

**global**

**Obslt**: subset of **TS** obsolete for regression testing

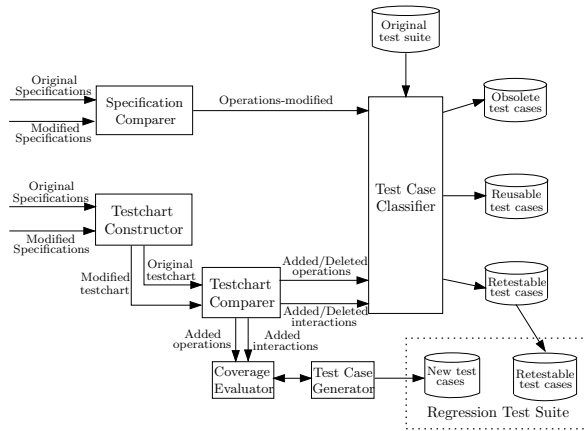
```

1.  Begin
2.    Obslt =  $\phi$    RTS =  $\phi$ 
3.    for each (ts  $\in$  TS) do
4.      get bit vector Operations-Used of ts
5.      for each ( $O_i \in \text{OpDel}$ ) do
6.        while ( $O_i^{\text{th}}$  bit of
7.          Operations-Used == 1) do
8.          Obslt = Obslt  $\cup$  {ts}
9.        endwhile
10.     endfor
11.     get bit vector Interactions-Used of ts
12.     for each ( $\text{IntrAct}_i \in \text{IntrActDel}$ ) do
13.       while ( $\text{IntrAct}_i^{\text{th}}$  bit of
14.         Interactions-Used == 1) do
15.         Obslt = Obslt  $\cup$  {ts}
16.       endwhile
17.     endfor
18.     for each (ts  $\in$  (TS - Obslt)) do
19.       get bit vector Operations-Used of ts
20.       for each ( $O_i \in \text{OpMd}$ ) do
21.         while ( $O_i^{\text{th}}$  bit of
22.           Operations-Used == 1) do
23.             RTS = RTS  $\cup$  {ts}
24.           endwhile
25.         endfor
26.       endfor
27.     endfor
28.  End

```

**Figure 4. Regression Test Selection Algorithm.**

problem [10]. The specification of a system is an ideal starting point for the systematic derivation of test cases. Therefore, specification-base test generation is an effective solution to the test suite augmentation problem. In TcozRts the new test cases are generated based on TCOZ specification; the process of test generation is guided by the information resulting from the comparison of the two versions of the



**Figure 5. TCOZ-based Regression Testing System: TcozRts.**

testcharts.

Figure 5 shows the architecture of the TCOZ-based regression testing system, TcozRts, and the interactions between the components of the system. *Specification comparer* takes original and modified versions of the specification and identifies the modified operations by automatically parsing the two versions of the TCOZ specification. The information about the modified operations is sent to the *test case classifier*. The *testchart constructor* constructs testcharts based on the two versions of the specification. The *testchart comparer* compares the two versions of the testchart to identify the added/deleted operations and the added/deleted interactions and sends the information about the modifications to *test case classifier*. With the information from *specification comparer* and *testchart comparer*, the *test case classifier* categorizes the original test cases as being obsolete, reusable or retestable. The retestable test cases will be used for regression testing. Since there are added operations and added interactions, *test generator* will generate new test cases for the testing of these new operations and interactions, in collaboration with *coverage evaluator*.

## 6. Related Work

Many techniques have been proposed for testing classes. For example, Doong and Frankl [9] developed ASTOOT which provides tools for automatic driver generation from a class interface specification and semi-automatic test case generation from algebraic class specifications. Ugo Buy *et al.* [4] proposed a framework that combines data flow analysis, symbolic execution and automated deduction in an effort to generate method sequences for structural testing of

classes.

For the regression testing of classes, there also have been a number of techniques developed. However, most of them are based on control flow or data flow analysis of code.

Rothermel, Harrold and Dedhia [24] presented a regression test selection technique for C++ software. The technique is strictly code based; it selects test cases, only using the information gathered by code analysis. Rothermel and Harrold also had presented a regression test selection for C++ software based on walks of program dependence graphs in [22]. However, the construction of program dependence graphs is more expensive than construction of control flow graph. Therefore, the technique presented in [24] is more efficient than which is presented in [22]. Harrod *et al.* [10] presented the first safe regression test selection technique that handles the features of Java language. Compared to Rothermel, Harrold and Dedhia's technique for C++ [24] and White and Abdullah's firewall technique [28], the technique presented in [10] is more precise, can be applied to incomplete programs, handles exception-handling constructs, and provides a new method for handling polymorphism.

As the counterpart of code-based regression testing techniques, only a few specification-based regression testing techniques have been developed. Korel *et al.* [12] presented a regression testing approach that uses Extended Finite State Machine (EFSM) model dependence analysis to reduce regression test suite. Briand *et al.* [3] presented a methodology and a tool support for automating safe regression test selection based on architecture and design information represented with the Unified Modeling Language (UML) and traceability information linking the design to test cases.

## 7. Conclusions

In this paper, we present a specification-based regression testing technique for classes specified in TCOZ notation. We mainly addresses the regression test selection problem and test suite augmentation problem involved in a typical selective retest technique. A TCOZ specification-based regression test selection algorithm is presented; it selects test cases related to the changed specification, from the original test suite, for use in the regression testing. We also present a TCOZ specification-based regression testing system, TcozRts, which implements our regression testing technique. The technique presented in this paper can be fully automated. It is strictly specification-based; no complex static or dynamic code analysis is required, therefore, it is independent of the programming language which is used to implement the specification.

## References

- [1] A. A. Andrews, R. B. France, S. Ghosh, and G. Craig. Test adequacy criteria for UML design models. *Software Testing, Verification and Reliability*, 13(2):95–127, 2003.
- [2] T. Ball. On the limit of control flow analysis for regression test selection. In *ACM International Symposium on Software Testing and Analysis*, pages 134–142, 1998.
- [3] L. C. Briand, Y. Labiche, and G. Soccar. Automating impact analysis and regression test selection based on uml designs. In *Proceedings of International Conference on Software Maintenance*, pages 252–261, 2002.
- [4] U. A. Buy, A. Orso, and M. Pezzè. Automated testing of classes. In *Proceedings of the ACM/SIGSOFT International Symposium on Software Testing and Analysis, ISSTA 2000*, pages 39–48, 2000.
- [5] D. Carrington, D. Duke, R. Duke, P. King, G. Rose, and G. Smith. Object-Z: An object-oriented extension to Z. In S. Vuong, editor, *Formal Description Techniques, II (FORTE'89)*, pages 281–296. North-Holland, 1990.
- [6] Y. F. Chen, D. S. Rosenblum, and K. P. Vo. TestTube: A system for selective regression testing. In *Proceedings of the 16th International Conference on Software Engineering*, pages 211–222, May 1994.
- [7] J. S. Dong, P. Hao, S. C. Qin, J. Sun, and Y. Wang. Timed patterns: TCOZ to Timed Automata. In *Proceedings of 6th International Conference on Formal Engineering Methods, ICFEM 2004*, pages 483–498, 2004.
- [8] J. S. Dong, S. C. Qin, and J. Sun. Generating MSCs from an integrated formal specification language. In *Proceedings of 4th International Conference on Integrated Formal Methods, IFM 2004*, pages 168–186, 2004.
- [9] R. K. Doong and P. G. Frankl. The ASTOOT approach to testing object-oriented programs. *ACM Transactions on Software Engineering and Methodology*, 2(3):101–130, 1994.
- [10] M. J. Harrold, J. Jones, T. Li, D. Liang, A. Orso, M. Penning, S. Sinha, S. Spoon, and A. Gujarathi. Regression test selection for Java software. In *Proceedings of the ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications OOPSLA 2001*, pages 312–326, October 2001.
- [11] P. Hsia, X. Li, C.-T. H. D. Kung, L. Li, Y. Toyoshima, and C. Chen. A technique for the selective revalidation of OO software. *Software Maintenance: Research and Practice*, 9:217–233, 1997.
- [12] B. Korel, L. H. Tahat, and B. Vaysburg. Model based regression test reduction using dependence analysis. In *Proceedings of the International Conference on Software Maintenance*, pages 214–223, 2002.
- [13] H. K. N. Leung and L. J. White. Insights into regression testing. In *Proceedings of the International Conference on Software Maintenance*, pages 60–69, 1989.
- [14] H. K. N. Leung and L. J. White. A cost model to compare regression test strategies. In *Proceedings of the Conference on Software Maintenance '91*, pages 201–208, October 1991.
- [15] H. Liang and J. Sun. Test Generation for Class-Level Testing from TCOZ Specifications. Technical report, School of Computing, National University of Singapore, 2004.
- [16] B. Mahony and J. S. Dong. Network Topology and a Case Study in TCOZ. In J. Bowen, A. Fett, and M. Hinchey, editors, *ZUM'98: The 11th International Conference of Z Users*, volume 1493 of *Lecture Notes in Computer Science*, pages 308–327, Berlin, Germany, Sept. 1998. Springer-Verlag.
- [17] B. Mahony and J. S. Dong. Overview of the semantics of TCOZ. In K. Araki, A. Galloway, and K. Taguchi, editors, *IFM'99: Integrated Formal Methods, York, UK*, pages 66–85. Springer-Verlag, June 1999.
- [18] B. Mahony and J. S. Dong. Timed Communicating Object Z. *IEEE Transactions on Software Engineering*, 26(2):150–177, Feb. 2000.
- [19] B. P. Mahony and J. S. Dong. Blending Object-Z and Timed CSP: An introduction to TCOZ. In K. Futatsugi, R. Kemmerer, and K. Torii, editors, *The 20th International Conference on Software Engineering (ICSE'98)*, pages 95–104, Kyoto, Japan, Apr. 1998. IEEE Press.
- [20] A. J. Offutt, Y. W. Xiong, and S. Y. Liu. Criteria for generating specification-based tests. In *Proceedings of the fifth International Conference on Engineering of Complex Computer Systems*, pages 119–131, Las Vegas, 1999. IEEE Computer Society Press.
- [21] J. Offutt, S. Y. Liu, A. Abdurazik, and P. Ammann. Generating test data from state-based specifications. *Software Testing, Verification and Reliability*, 13:25–53, 2003.
- [22] G. Rothmel and M. J. Harrold. Selecting regression tests for object-oriented software. In *Proceedings of the Conference on Software Maintenance*, pages 14–25, 1994.
- [23] G. Rothmel and M. J. Harrold. A safe, efficient regression test selection technique. *ACM Transactions on Software Engineering and Methodology*, 6(2):173–210, April 1997.
- [24] G. Rothmel, M. J. Harrold, and J. Dedhia. Regression test selection for C++ software. *Journal of Software Testing, Verification, and Reliability*, 10(6):77–109, June 2000.
- [25] S. Schneider and J. Davies. A brief history of Timed CSP. *Theoretical Computer Science*, 138, 1995.
- [26] J. Sun, J. S. Dong, J. Liu, and H. Wang. A XML/XSL Approach to Visualize and Animate TCOZ. In *The 8th Asia-Pacific Software Engineering Conference (APSEC'01)*, pages 453–460. IEEE Press, 2001.
- [27] F. Vokolos and P. Frankl. Pythia: A regression test selection tool based on textual differencing. In *International Conference on Reliability, Quality, and Safety of Software Intensive Systems*, May 1997.
- [28] L. J. White and K. Abdullah. A firewall approach for regression testing of object-oriented software. In *Proceedings of 10th Annual Software Quality Week*, May 1997.
- [29] L. J. White and H. K. N. Leung. A firewall concept for both control-flow and data-flow in regression integration testing. In *Proceedings of the conference on Software Maintenance '92*, pages 262–270, November 1992.