Contents lists available at ScienceDirect

Advances in Engineering Software

journal homepage: www.elsevier.com/locate/advengsoft

A safe regression testing approach for safety critical systems

Zahid Hussain Qaisar [a,*], Shafiq Ur Rehman [b,*]

[a] *Faculty of Information Technology, University of Central Punjab, Lahore, Pakistan*
[b] *Faculty of Computing, Riphah International University, Islamabad, Pakistan*

ARTICLE INFO

Article history:
Received 14 December 2010
Received in revised form 21 April 2011
Accepted 23 April 2011
Available online 19 May 2011

Keywords:
VDM specification
Regression testing
Test suite selection
Formal specification based regression testing
VDM specification based testing
Changed identification
Test suite selection for regression testing

ABSTRACT

Regression testing is important activity during the software maintenance to deal with adverse effects of changes. Our approach is important for safety critical system as usually formal methods are preferred and highly recommended for the safety critical systems but they are also applied for the systems development of other than critical system. Our approach is based on Regression testing using VDM++ which takes two VDM++ specifications, one baseline and other delta (Changed) along with test suite for the baseline version. It compares both versions by using comparator module, identifies the change. By analyzing the change we classify the test cases from original test suite into obsolete, re-testable, and reusable test cases. Our scope is at unit level i.e. at class level. Our approach gets two versions of VDM++ specification and returns regression test suite for the delta version. Our approach distinguishes test cases which are still effective for the delta version of VDM++ specification and it differs from re-test all strategy as it can distinguish the test cases and identifies test cases which are useful for delta version. Test cases reusability and test case reduction is the main objective of our approach. Our approach presents how to perform regression testing using VDM++ specification during the maintenance of systems.

1. Introduction

During software development, maintenance phase is very expensive as most of time is spend on maintenance phase when we require modification in the requirements [1]. However it is extremely important to deal with the changes introduced during the maintenance phase. Regression testing is an activity to ensure that these changes have been made, tested and has no unintended affect on the system after the maintenance. Improvements in existing regression testing techniques and developing new methodologies are very important to build robust systems [2].

Existing literature about VDM++ is lacking approach for regression testing of VDM++ specification as most of work has been done on test case generation. VDM++ is the object oriented specification language just like Z for the formal specification of the systems. It is different from the Z in structure as it separately has pre and post condition which are lacking in Z. Pre and post condition of predicate are important in test oracles where we use them to test the system [3]. We have chosen VDM++ specification as it is object oriented version of VDM which is mostly used now a days in development of safety critical systems.

The maintenance is categorized into three types, Corrective maintenance, Adaptive maintenance and Perfective maintenance. The changes made in each type of maintenance have different impacts on the software, that is why regression testing is based on the kind of maintenance performed [4]. Corrective maintenance is such type of maintenance in which some correction is made in the software to correct the errors [5]. Adaptive maintenance is performed to make the software compatible for new environment and Perfective maintenance is performed when there is a requirement of addition of new modules. In first kind of maintenance, requirements do not change while in the second and third kind of maintenance requirements have to be changed. Regression testing types are based on whether the requirements are changed or not [6].

Regression testing can be experienced in three basic types, Progressive regression, Adaptive regression and Corrective regression. First and second type of regression testing is applied when the requirements specifications are modified while the third type of the regression testing is applied when the specification does not change but we need some correction in the software. Possibly some error is removed or may be any efficient algorithm is implemented replacing the old one due to any reason [1]. The increasing use of formal methods in software industry has increased the importance of formal specification based testing techniques. Though formal methods are increasingly used in the software industry but still there is very less work done as far as the formal specification based testing is concerned. In this paper, we present an approach for the systems specified in VDM to find the impact of change in the specification on the test suite. Our approach identifies changes in two versions of VDM specification; the baseline

* Corresponding author.
E-mail addresses: zahidqaisar@yahoo.com (Z.H. Qaisar), shafiq.rehmaan@gmail.com (S.U. Rehman).

0965-9978/$ - see front matter © 2011 Elsevier Ltd. All rights reserved.
doi:10.1016/j.advengsoft.2011.04.007

and the delta version. After the change identification, impact of changes on the test suite is analyzed for regression test selection. The rest of paper is organized as follows: Section 2 describes the related work. Section 3 discusses our regression testing approach in detail. Section 4 summarizes the results of case study, Section 5 discusses the implementation issues and Section 6 concludes the approach.

2. Related work

A lot of work has been done for the specification based testing but a less work is done for the specification based regression testing. We have analyzed all work related specification based testing as well as specification based regression testing. Most of the work is done for the Z specification based testing but for VDM specification less work is done.

Technique proposed by Baydeda and Gruhn is about class specification and implementation graph and their application in regression testing [7]. This technique is not appropriate for the VDM Specification and has overhead of generating graph from the specification. This technique also lacks coverage criteria as it just provides branch coverage criteria to some extent.

Harbhajan Singh has done work on formal specification testing by using Z language and has proposed test case design based on Z and using classification tree [8]. Amyerah and Zin has also done work on Z for formal specification based testing but Dorsch and Nadeem has done work on VDM specification based testing. Dorsch has proposed design and application or test generator for VDM-SL [9] which can be suitable for our proposed approach for test suite generation of baseline version specification in our context but most appropriate is work of Nadeem for our approach as it considers both specification and implementation and consider pre and post condition of operation for test cases generation from VDM specification so we proposed that for integration with our proposed approach TESTAF approach is more suitable[10].

For formal specification based regression testing there is not sufficient literature as little work has been done and most of work is related to other specification languages rather than VDM specification. Chakarabarti work is related to specification based regression testing but is used finite automata for the comparison of two versions of specifications and for change identification so it is related to model i.e. model based [10.SimilarlyHeimedalandGeorgeworkalsousesmodel [11]. Liang work is related to object-Z and is not for VDM specification. It is for classes of Object-Z [5]. Charabarti has also done work regression testing of Reactive systems and an approach is given for test sequence computation of reactive systems [11]. Marre and Blanc work is for the Luster specification language they have proposed test selection strategies for the luster specification language. Beydeda and Gruhn work is for class specification and implication graph for the change identification of Delta version specification which is more complex as it uses both specification and implementation secondly it is not for the VDM specification. Work by Fraster and some other authors work for the model based regression testing or for the component based regression testing which is not even specification based regression related so from this related work we can clearly see that there is need for the VDM specification based regression technique as during the maintenance phase when there is change we can identify change and by analyzing this change we can select appropriate test suite more effectively and efficiently. VDM is widely used for the specification of different systems and there is VDM tool which provide facility for the automatically generation of code either for java or for C++ directly from the specification therefore implementation and specification comparison is not much important. So in next section we will discuss an approach for VDM specification based regression testing which can be integrated with already existing TESTAF approach for test suite generation for the baseline version specification and by using our proposed approach will take two VDM specifications for comparison and will define mechanism for change identification and appropriate test suite selection.

3. Proposed approach

We propose an approach for regression testing which is based on formal specification of VDM-SL language. Input will be formal specification document written in VDM++ (Vienna Development Methods) which is a formal specification language. We analyze the formal specification developed in VDM. We compare the unchanged version of the specification (baseline specification) with the changed version of the specification, also called delta version of the specification. This comparison is based on the syntactic properties of the specification. After performing the syntactic comparison, we also compared our specification semantically to uncover the predicates which are semantically equivalent as predicate may syntactically different but semantically same [10]. For example

$a > b$ is semantically equivalent to $b < a$

Therefore our approach checks both syntax and semantics of both the specifications. As mentioned in the Fig. 3.1 our approach will take two versions of specification as input. Pre-condition and post condition of operation are extracted by pre and post condition extractor of both baseline and delta versions of specification. A precondition of our approach is that pre and post conditions of the operations in each VDM++ class should be in the disjunctive normal form (DNF) [3]. However, this is not a hard pre-condition as every predicate can be transformed into DNF form by applying.

We build regression test suite by comparing pre and post conditions of operations on the basis of baseline version of the specification. Test selection algorithm is designed for selection of test cases.

3.1. Parsing of VDM specifications

In parsing of VDM++ specification we parse the baseline and delta VDM++ specification. We get parse tree and through parse tree we get pre and post condition and variables list for baseline specification and delta specification. Later on we use them to compare both the specifications and identify change by comparing both specifications. Given below is algorithm to parse the baseline version of VDM++ specification and Delta version specification.

3.2. Specification comparator

In our proposed approach we compare both the specifications i.e. Baseline specification and delta version specification of VDM++. Comparison of both the specification tells that whether the delta specification has changed or not and if change has been done what is the mechanism for the test suite selection for the regression testing. Output of the parser mentioned above in Fig. 3.1 is the input of specification comparator. We have given algorithm for the comparison of specifications which compares the specification syntactically and semantically. VDM++ specifications will be parsed and their pre and post conditions of the predicates are stored in separate lists. Then comparator will get this information from the lists and compares them. First comparison will be string comparison if change identified then it will semantically check whether this change is just syntactic change or it also has effect on semantics. For Semantic comparison we have defined

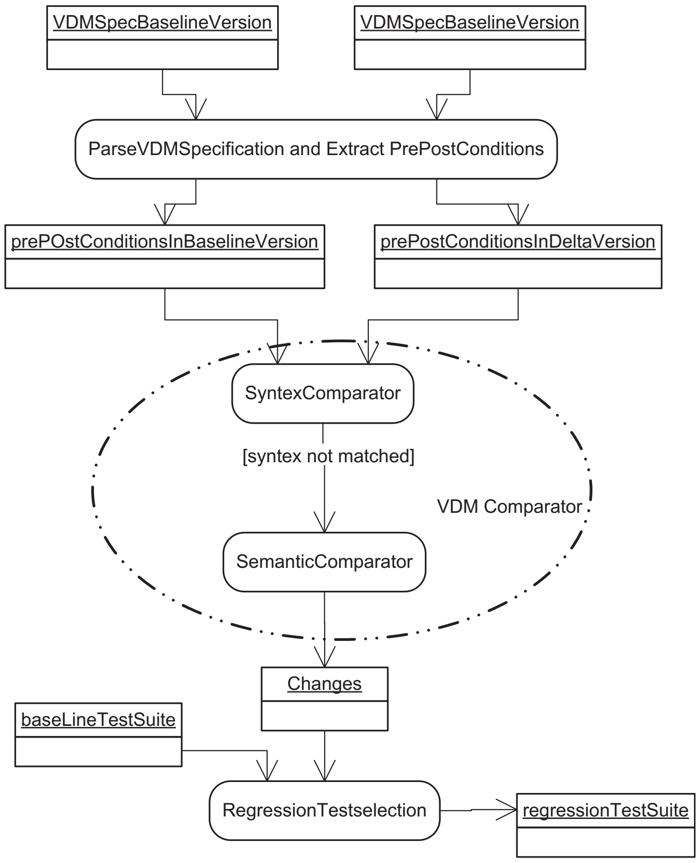

Fig. 3.1. Architecture of approach.

equivalent rules for the each operator in VDM++ which defines that when the two predicates are equivalent and in which cases they differ semantically. Comparison is based on the output of the parser. Parser gives us the parse tree which is the input of the comparator module.

3.3. Pre-condition and post condition comparison

We evaluate pre and post conditions of the specification as these are the change identification points i.e. where we can determine the change in the previous and the new version of the specification written in VDM++. This comparison is on two bases, syntactically and semantically. Pre-condition is the filter for the test input and post condition is what the operation should do after end of that operation i.e. result of the operation is post condition while the condition that should be satisfied before the entry to operation.

3.4. Syntactical and semantic comparison

Comparison of pre and post condition was both in term of syntax and semantics in our proposed technique. As just syntactical comparison does not guarantee the actual change in the delta version of the specification so we have also provided the mechanism to check the semantics of the pre and post conditions of the specification. Semantic comparison is the key mechanism to detect the change and its level.

3.5. Syntactical comparison

First of the all we compare the two versions of specification syntactically and find change in the syntax of new version. But even change in syntax does not guarantee change in the test suit as there is possibility that specification might have different syntax but semantically have the same impact on the test suit.

3.6. Semantically comparison

As we have mentioned in syntax comparison only syntax comparison does not guarantee that change in syntax will lead to change in test suite. So, we have to also check specification semantically. In semantic comparison we check keeping in mind that different syntax can lead to similar semantics. As there are some operators in the VDM++ specification language that can cause change in the semantics of the specification i.e. operands position is important if we change the operands position semantics will be different. On the other hand there are also operators which do not lead to change in the semantics of the specification and eventually have no impact on regress suit if operands position is changed.

3.7. Algorithm for comparator and test case selector

For our approach operation or function’s ID (identity) should be unique and in this algorithm we are assuming that original Specification of VDM++ before change is Baseline VDM++ specification (Baseline Version) and after change is Delta version VDM++ specification (Delta Version).

In our approach pre and post condition of operations are matched or compared with operations of Delta version syntactically as well as semantically and a list of changed and unchanged operations are maintained, changed operations are placed in the operation changed list. Pre and post condition of operation are extracted by the parser of VDM++ specifications mentioned Fig. 3.1 which extracts and passes it to specification comparator.

In our algorithm output of parser is the input of the comparator. Comparator compares both versions of specification and on basis of comparison change is identified in the classes and operations of the delta version. Changed operations are stored in the changed operation and unchanged operations are stored in the unchanged operations list. Comparator at the end returns list of changed, unchanged and deleted operations and then these lists are used by the test selector. Test selector selects test cases for the unchanged operation i.e. selects re-useable test cases for the unchanged operations of delta version from the list of test cases for the baseline version.

For changed operations list test cases selector do not select the test cases and it puts corresponding test cases of the operations in the list of obsolete test cases. Similarly comparator also identifies the deleted operation and deleted classes as well as new operations and new classes, for these operations and classes test case selector puts corresponding test cases in the obsolete test cases list. Similarly for the changed operations test case selector will not select the test cases and will put them in the obsolete test cases list i.e. these are not re-useable test cases and for these kind of operation we have to make new test cases to test them i.e. re-testable test cases.

3.8. Implementation of our proposed approach

As far as implementation of our proposed approach is concerned we integrate it with existing tools for the testing of VDM++ specification. For baseline version of specification we can get test cases by using TESTAF approach [10]. After getting test cases for the baseline version of the specification we compare two specifications i.e. baseline version specification and Delta version of specification. After analyzing these changes we will select appropriate test cases from the baseline version test suite for the regression test suite.

As we can see the association between the classes used in the algorithm for comparator and test cases selector from class diagram in the given below diagram. Fig. 3.2 represents class diagram of our tool.

Our proposed approach can be integrated with the TESTAF [10] framework at test case generator component of that framework for the regression test suite selection. For getting the test cases for the baseline version we can apply any existing approach but we are planning to integrate it with TESTAF framework. Our approach will be used for test selection for the Delta version of specification i.e. regression test suite. We can execute these test cases on the Delta version specification by using TESTAF approach which provides facility for test cases execution automatically through test drivers. So; our proposed approach can easily be integrated with TESTAF approach which is basically a test automation framework for class testing using object oriented formal specification i.e. VDM++ specification based testing. Test cases are automatically generated by using this approach and after test suite generation our proposed approach is integrated at its test case generator component for the test selection for the Delta version of the specification. After test selection of regression test suite by our proposed approach, test cases can be automatically executed by using test drivers proposed of TESTAF approach.

3.9. Types of possible changes in the specification

Following are possible changes in the specification that may lead change in the regression test suite.

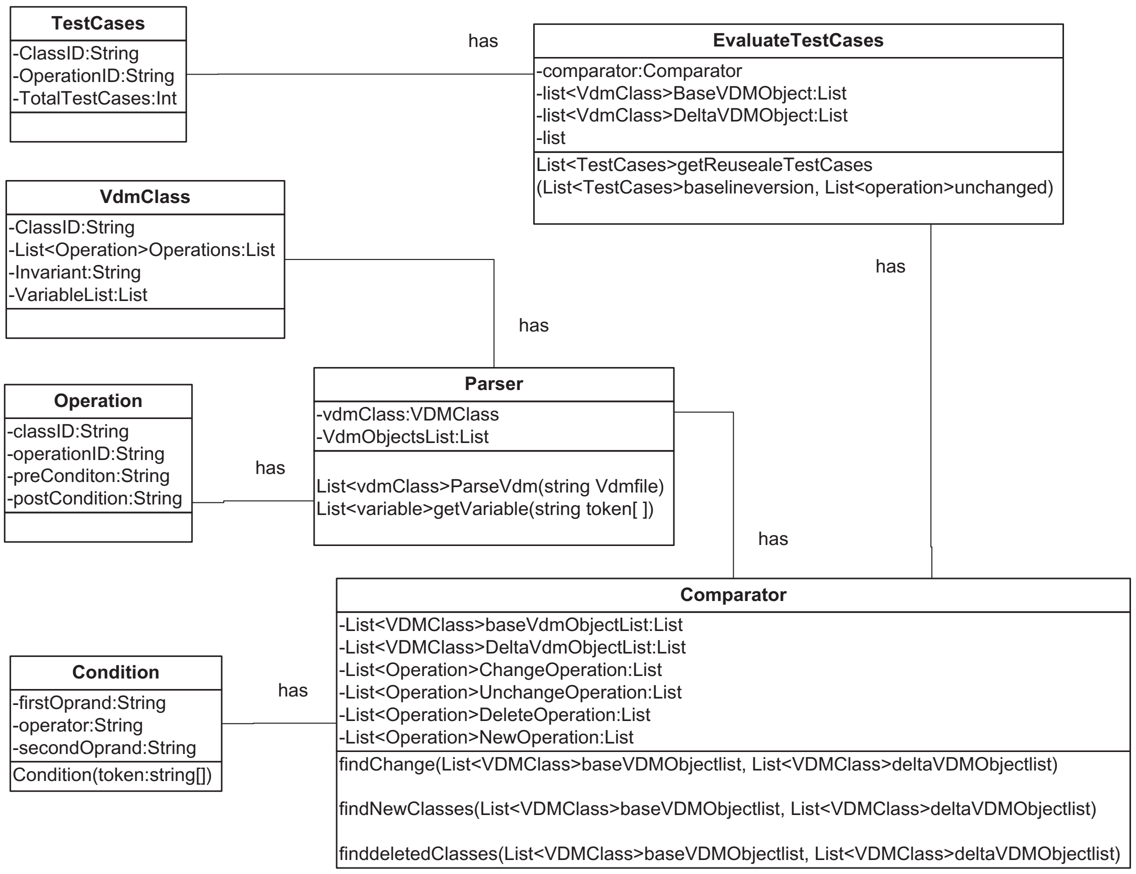
TestCases
-ClassID:String
-OperationID:String
-TotalTestCases:Int
has
EvaluateTestCases
-comparator:Comparator
-list<VdmClass>BaseVDMObject:List
-list<VdmClass>DeltaVDMObject:List
-list
List<TestCases>getReusealeTestCases
(List<TestCases>baselineversion, List<operation>unchanged)
VdmClass
-ClassID:String
-List<Operation>Operations:List
-Invariant:String
-VariableList:List
has
has
Operation
-classID:String
-operationID:String
-preConditon:String
-postCondition:String
has
Parser
-vdmClass:VDMClass
-VdmObjectsList:List
List<vdmClass>ParseVdm(string Vdmfile)
List<variable>getVariable(string token[ ])
has
Comparator
-List<VDMClass>baseVdmObjectList:List
-List<VDMClass>DeltaVdmObjectList:List
-List<Operation>ChangeOperation:List
-List<Operation>UnchangeOperation:List
-List<Operation>DeleteOperation:List
-List<Operation>NewOperation:List
findChange(List<VDMClass>baseVDMObjectlist, List<VDMClass>deltaVDMObjectlist)
findNewClasses(List<VDMClass>baseVDMObjectlist, List<VDMClass>deltaVDMObjectlist)
finddeletedClasses(List<VDMClass>baseVDMObjectlist, List<VDMClass>deltaVDMObjectlist)
Condition
-firstOprand:String
-operator:String
-secondOprand:String
Condition(token:string[])
has

Fig. 3.2. Class diagram of proposed approach.

3.9.1. Changes in a VDM class

In VDM++ specification a class can have operations, pre and post condition etc. [12]. So, we have followed that definition that pre, post and operations can be changed. We have analyzed that when we change specification from baseline to delta version, what kind of impact it can cause in term of specification based testing.

Change in the delta version class can be either addition of new class, modification or deletion of class. Change in name of class leads to re-testable test cases. Similarly identifier change leads to change in class ID in the delta version which will be identified by syntax checking in our approach. The deletion and the addition of classes in the delta version specification will lead to obsolete or re-testable test cases. In case of addition of new classes all test cases for that class will be considered as re-test able test cases similarly in case of deletion of class all operations of that class will be deleted which will lead obsolete test cases of all related operations and corresponding class.

Change in inheritance clause leads to change in the other classes in the delta version as well. If the changed class was super class i.e. parent class it lead to change in the child classes inherited from this parent class. At present, in our approach we are not dealing with the inheritance of classes. Change in the body of a class is caused due to change in the operations or functions or change in the state variables. We will discuss these changes in detail in the next sections.

Above mentioned Class diagram in Fig. 3.2 shows the association between different modules of our technique. As earlier we have mentioned the sequence of our technique. In class diagram, we can view the association among the different modules i.e. Comparator, parser, Test selection and Input modules of our technique.

3.9.2. Changes in operations

In VDM specification, an operation is defined as implicit, explicit or internal [12]. Implicit operations are the operations which do not have body and just have pre and post condition along with externals. In standard VDM we do not have extended explicit operations as these are allowed only in VDM tool provided by CSK Corporation from which we have taken this definition. In implicit operations and functions we just have pre and post conditions and do not have operation body i.e. we do not know how to get post condition. While in case of explicit operation we have pre and post condition along with operation body which describes that how to achieve this post condition.

In case of explicit operation, we have to keep consideration of body of the operation along with the pre and post condition. Change in the body of operation means change in the how part i.e. how the post condition will be achieved will be changed. In delta version of specification change in body part of operation may or may not lead to change in the test cases but change in the post condition will definitely cause change in the test cases for the delta version of the specification.

Our approach works well for the implicit operations and can identify all changes in the implicit operation however in case of explicit it cannot include such cases where body of operation has been changed and this change have no effect on pre and post condition of operation (exceptional case) but this is very rare case as usually body is a way of implementation and a major change will lead to either change in pre or post condition.

Externals are also important in the operations as change in the externals can change the test cases i.e. change in the data type or change in the variable will cause change in the test cases. Similarly exceptions can also play role in the regression testing as exceptions tell that what should be the behavior of system in case of that particular condition which can occur in the system so change in the exception will cause changes in the regression testing. Basically exception tells what should be the system response in case of a particular exception when it occurs if we found change in the exception our test cases will be different.

In case of change in pre and post condition the test cases for the operation will be changed and in our proposed approach we will cater them. If change is in externals it will also lead change in test cases as in externals we mention data types and variables so our approach is dealing with the change in the data type as well as change in the number of variables.

Comparison of pre and post conditions of operations has significant importance in our approach. As pre and post condition are basically entry and exit point of the operations and can play better role in change identification. In our approach we compare pre and post conditions of operations as these are change identification points.

In semantic comparison of specification we check change in operations by comparing pre and post condition of the operation. Pre and post condition are predicates basically so change in operation's pre and post condition will be change in the operations.

Addition of operation will occur when we want to add some functionality in the new version of the specification which was not being provided in the previous baseline version of specification. For example our old specification was providing arithmetic operations like addition and subtraction to be performed on natural numbers in the new version we want to add functionality of multiplication so we have to add multiplication operation in Delta version. Addition of operations will lead to the change in regression test suite by addition of new test cases while no change will lead to re-testable test cases in the specification i.e. test cases will remain same as for the previous specification and are still re-useable, these test cases will be selected by the test selector.

Deletion will occur when we no longer require functionality in new version of specification that was being provided in the older version of specification i.e. for example in above-mentioned example we do not want functionality of subtraction in new specification so we have to delete subtraction operation. Deletion of operation will lead to obsolete test cases in regress test suite. Similarly modification in test cases will cause re-testable test cases addition in the test suit. Hence test cases for unchanged part of specification will be categorized in the re-useable test case category.

3.9.3. Changes in variables

In the new version of specification or in the changed version of specification either there can be addition of variable, deletion or modification of variable by data type change. Of variable or change in name of variable. In semantic checking, we ignore name of the variable i.e. variable with different name may have same functionality as in the previous version that name was different. When there is change in the variable either it is addition of variable that is identified by pre and post condition check at the operation level. So, deletion and addition of variables in the specification will definitely lead to change in the regression test suite. Test case selector will select test cases and for that particular change for example for deletion of variable test case selector will select or identify obsolete test cases. The test selector will do test case selection.

Change in variable will occur according to our requirements for example a variable will deleted when we find that it is no longer required and our specification will even provide functionality required after deletion of this variable i.e. it was additional variable not necessarily required in the older version of specification so deleted in the new version. Similarly a variable will be added when we require some values to provide functionality from specification i.e. additional variable is required to provide some required values for the Delta version of specification so variable will be added.

In next section of result and discussion we have elaborated and illustrated the concept of addition and deletion of variables with the help of case study. Change in data type will be deal as change in variable. For example if we change data type of variable from real to nat type this will be deal as variable change and will be compared syntactically and semantically by the comparator.

Deletion of variable will cause generation of obsolete test cases while addition will cause re-testable test cases. Similarly modification will cause re-testable and re-useable test cases as mentioned above example, if baseline version has variable of type real and in delta version its data type has changed from real to nat then test cases for real type will become obsolete and we will require new test cases for the nat type.

3.10. Format of baseline test suite

The baseline test suite should consist of the test cases for each VDM++ class and its corresponding operation i.e. each operation of the class have separate test cases with operation and class ID (identity) to make them traceable to the specification. Test suite has been shown in Table 1.

3.10.1. Test suite selection

By comparing two versions of specification the baseline version and delta version by pre and post condition comparator we will select test cases for the delta version. Base for the test selection for the delta version will be the test suite of baseline version. So after comparison we will select regression test suite for the delta version of the specification. If the change is syntactical and there is no impact on semantic then reusable test cases will be selected and if the change is semantic or such syntactical changes that have impact on semantics it lead to re-testable test cases as mentioned in the case study.

The test case selector will do test suite selection after pre and post condition evaluation of the operation. For test suite selection base will be the database of test cases for baseline version. Test selector will select test cases according to the identification of change in the Delta version of VDM specification. After all test cases, selector will give regression Test suite.

3.10.2. Obsolete test cases

When we have generated test suite from the baseline specification for the delta version we have to classify the test cases by comparing the two versions of the specification the old version without change and the new version which has evolved after the change. After analyzing and comparing the two versions of specification specified in VDM++, we came to know about some cases that were valid for old specification are no longer required for the new version of the specification. i.e. test cases are now obsolete after the change in the specification. For example there is a variable whose data type is real specified in the older version of the specification and in the new version the data type is changed and new data type specified for that variable is real type so test cases that were for real data type and formed for testing this type is no longer be required for nat data type and becomes obsolete for the changed version of the specification. Such test cases are classified as obsolete test cases and are kept in the obsolete category of the test cases.

3.10.3. Re-testable test cases

Test cases for the modified part of the specification are called re-testable test cases. These are such test cases, which are specific to the modified part of the specification and are not for the previous or older version of the specification. These are designed to test the modified part of the specification or the changed part of the specification.

In case of changing data type from Nat to real data type [19–21]. Test cases specific to test real data type are re-testable test cases and are categorized in the re-testable test case in testing.

Table 1
Test cases for baseline versions.

Test case no.	Input values			Expected output	Test case no.	Input values			Expected output
Test cases for add operation					*Test cases for subtract operation*				
1	re = 0	im = 0	$x = 0$	0.0	16	re = 0	im = 0	$x = 0$	0.0
2	re = 0	im = 0	$x = 1$	1.0	17	re = 0	im = 0	$x = 1$	−1.0
3	re = 0	im = 0	$x = 9$	9.0	18	re = 0	im = 0	$x = 9$	−9.0
4	re = 0	im = 1	$x = 0$	0.1	19	re = 0	im = 1	$x = 0$	0.1
5	re = 0	im = 1	$x = 1$	1.1	20	re = 0	im = 1	$x = 1$	−0.9
6	re = 0	im = 1	$x = 9$	9.1	21	re = 0	im = 1	$x = 9$	−8.9
7	re = 0	im = 8	$x = 0$	0.8	22	re = 0	im = 8	$x = 0$	0.8
8	re = 0	im = 8	$x = 1$	1.8	23	re = 0	im = 8	$x = 1$	−0.2
9	re = 0	im = 8	$x = 9$	9.8	24	re = 0	im = 8	$x = 9$	−8.2
10	re = 1	im = 0	$x = -1$	0.0	25	re = 1	im = 0	$x = -1$	0.0
11	re = 1	im = 1	$x = 0$	1.1	26	re = 1	im = 1	$x = 0$	1.1
12	re = 1	im = 8	$x = 12$	13.8	27	re = 1	im = 8	$x = 12$	−10.2
13	re = 5	im = 0	$x = -5$	0.0	28	re = 5	im = 0	$x = -5$	10.0
14	re = 5	im = 1	$x = -4$	1.1	29	re = 5	im = 1	$x = -4$	9.1
15	re = 5	im = 8	$x = 6$	11.8	30	re = 5	im = 8	$x = 6$	−0.2
Test cases for multiply operation					*Test cases for divide operation*				
31	re = 0	im = 0	$x = 0$	0.0	46	re = 0	im = 0	$x = 0$	Exception
32	re = 0	im = 0	$x = 1$	0.0	47	re = 0	im = 0	$x = 1$	0.0
33	re = 0	im = 0	$x = 9$	0.0	48	re = 0	im = 0	$x = 9$	0.0
34	re = 0	im = 1	$x = 0$	0.0	49	re = 0	im = 1	$x = 0$	Exception
35	re = 0	im = 1	$x = 1$	0.1	50	re = 0	im = 1	$x = 1$	0.1
36	re = 0	im = 1	$x = 9$	0.9	51	re = 0	im = 1	$x = 9$	0.0
37	re = 0	im = 8	$x = 0$	0.0	52	re = 0	im = 8	$x = 0$	Exception
38	re = 0	im = 8	$x = 1$	0.8	53	re = 0	im = 8	$x = 1$	0.8
39	re = 0	im = 8	$x = 9$	7.2	54	re = 0	im = 8	$x = 9$	0.1
40	re = 1	im = 0	$x = -1$	−1.0	55	re = 1	im = 0	$x = -1$	−1.0
41	re = 1	im = 1	$x = 0$	0.0	56	re = 1	im = 1	$x = 0$	Exception
42	re = 1	im = 8	$x = 12$	21.6	57	re = 1	im = 8	$x = 12$	0.2
43	re = 5	im = 0	$x = -5$	−25.0	58	re = 5	im = 0	$x = -5$	−1.0
44	re = 5	im = 1	$x = -4$	−20.4	59	re = 5	im = 1	$x = -4$	−1.3
45	re = 5	im = 8	$x = 6$	34.8	60	re = 5	im = 8	$x = 6$	1.0

3.10.4. Re-useable test cases

These are the test cases which are useful for the changed version of the specification as well as for the old version of the specification. These test cases are valid for the old version of the specification as well as valid for new version of the specification after the change i.e. these test cases must be executed for both versions of the specification before and after the change.

For example if there is no change in the operation or class then all test cases for that particular operation is still valid i.e. test cases for the baseline version are still valid for the delta version if there is no change in the operations or classes in the delta version.

4. Case study

In this section, we present a case study as mentioned in Table 1 to verify our proposed technique. It is basically a class for complex numbers which has to perform the following four arithmetic operations addition, subtraction, multiplication and division of numbers. That example is for complex numbers how to add, subtract, divide and multiply complex numbers [13]. We have shown baseline and then delta version of our specification. Delta version of our specification contains some changes like data type changes, addition of variables, deletion and modification of operations. In this example for regression testing we have generated test cases for the baseline version and then selected test cases for the delta version of the specification i.e. regression test suite[14–16].

We have the baseline VDM++ specification as input on the basis of this specification version we find out test cases for the delta version of the specification and apply our approach on this case study. We compare pre and post conditions of operations in both versions of the specification and generate regression test suite for the delta version specification. Now we already have test cases for this baseline version as mentioned in Table 1.

Table 1 shows test cases for the delta version VDM++ specification by using these test cases we decide or evaluate these test cases for the delta version specification. Comparator in our approach identifies the change in the delta version and operation, if changed, is placed in the changed operations list and unchanged operations will be placed in the unchanged operations list. For our case study we find what kind of changes have been taken place. On basis of this change what are the test cases for the delta version specification i.e. re-useable and obsolete test cases [17,18,3] are defined for the delta version VDM++ specification.

4.1. Specification after change (Delta version)

Following are the changes that occurred in the modified part of the specification.

4.1.1. Data type change in subtraction operation

Test cases for boundary value analysis of the baseline version of the specification mentioned in Fig. 3.1 are as following.

Suppose for the Delta version of the above mentioned baseline specification of the baseline version we found following changes and we observed data type change in variable of the baseline version. So for this kind of change, test cases of baseline version for this operation (starting from 15 to 30) become obsolete, as nat 1 data type do not include zero in valid input space so these test cases are not valid test cases. Secondly for the change data type we require new test cases called re-testable test cases as changed data type causes change in the boundary values. Our test cases for boundary value analysis are changed and we have new test cases for this changed operation. Data type change in the operation in our proposed approach we put operation in the list of changed operations. We have to test new test cases i.e. re-testable test cases for the delta version of specification.

4.1.2. Syntactical change with no impact

In this example, we have found change in the comparator operator of addition method which lead in pre-condition change but this change is syntactical change which has no impact on semantic so no modification in test cases as in this case operator changed is comparator operator which has been classified in the change impacting operators i.e. which changes semantics of pre and post condition. If change is in their operands positions then this has been mentioned in our algorithm.

For that change in the specification all 15 test cases remain valid after the change and are categorized in the re-useable test cases database of test cases. Change detected by the syntax comparator i.e. there is syntax difference of operations of baseline version and delta version of specification.

After change identification of the operation syntactically our approach checks it semantically that whether this change is semantically different or not. In our case mentioned above change is syntactical which has no impact on the semantics of the operation so operation is added to the unchanged operation list by the semantic checker in our proposed approach. All test cases of that operation are same as for the baseline version and test cases for that operation are selected by the regression test selector [22,19,23,24].

4.1.3. Syntactical change with impact of change on semantics

Given below is example of syntactical change in the operation which has also impact on the semantics of the operation. Change in comparison operation changes the semantics of the operation.

In this case 15 test cases for the subtraction operation (starting from 30 to 45) mentioned above become obsolete while new cases i.e. re-testable are added to add this changed operation as changed operation is treated as a change in operation by our pre and post condition comparator and operation is added to list of changed operations which will require new test cases for the testing. If suppose there was only this change in operation of subtraction then all test cases for the other operation of the specification will still be effective for this delta version i.e. will be re-useable test case.

4.2. Implementation

We have developed a tool to illustrate our proposed approach. Our tool get test suite for the baseline version along with baseline and delta version VDM++ specifications as input. Test cases for the baseline version can be generated using TESTAF approach [10]. Our tool compares Baseline and delta version specification. Comparison is made after both specifications have been parsed i.e. parser parses both specification and then required information i.e. pre and post conditions of operations, class name, operations name and variable types and names are stored in the lists. After parsing, parser passes this information to the comparator. Comparator compares these lists in two ways first of all string comparison is made after this if two strings one from baseline version and other from delta version do not match then a semantic comparison is made which is based on the rule based system depending upon operator used in the pre or post condition of the operation.

After comparison made by the comparator the identified changes are passed to the test selection module which determines the test cases for the delta version specification. Test suite selection module select test cases from the baseline version and categories them in three categories i.e. re-testable, reusable and obsolete as mentioned above in detail.

Table 2
Results of case study.

Baseline version operations	Number of test cases for baseline	Changes identified	Number of test cases for baseline

Addition operation	15	Addition operation changed (semantic change)	Obsolete = 15
			Re-useable = 0
			Re-testable = 15

Subtraction operation	15	Subtraction operation changed (semantic change)	Obsolete = 15
			Re-useable = 0
			Re-testable = 15

Multiplication operation	15	Multiplication operation unchanged (syntactical change but no impact on semantics)	Obsolete = 0
			Re-useable = 15
			Re-testable = 0

Division operation	15	Division operation unchanged (no change)	Obsolete = 0
			Re-useable = 15
			Re-testable = 0

		Total = 60	
Total obsolete = 30			
			Total re-useable = 30 Total re-testable = 30
Total test cases to be executed for the Delta version = 30 reusable +30 re testable = 60 test cases			

Our tool can be integrated with the TESTAF approach for the generation of baseline version test cases. Our tool return regression test suite to test the baseline version VDM++ specification.

5. Evaluation

In above-mentioned case study we have shown that our proposed approach works well for these kinds of changes observed in the delta version of the specification. For boundary value analysis of baseline version of the specification we are having 108 No. of total test cases. Suppose if changes mentioned above have been observed in the Delta version of the specification then overall regression test suite for the delta version will be as mentioned in Table 2.

So as we have shown in the above-mentioned example that our approach has reduced number of test cases for the regression test suite. Our approach is different from re-test all strategy for the regression testing as it can distinguish obsolete and re-useable test cases for the delta version [25]. So we can claim that our approach is better approach as compare to re-test all and is more effective in test suite reduction.

Our approach is for VDM++ which is more common in use for the formal specification of highly critical systems. It is more preferred on Z as we know in Z both pre and Post conditions are embedded in a single predicate and we cannot easily separate them. However in VDM we have clear separation of pre and post conditions which is more beneficial in term of finding test cases [26–29]. So for testing VDM specification is more appropriate as we can easily generate test cases while in Z we normally rely on formal proof which is not sufficient or does not guarantee that our software will be bug free [11].

6. Conclusion

In this research paper, we have presented an approach for the regression testing by using VDM specification and our scope was at unit level i.e. at class level. We have presented a novel approach for regression testing which identifies the changed and then by analyzing that change [30,31] we will select test cases from the test suite of the baseline version specification. Change identification is at the class and operation level which caters deletion, addition and modification of the class and operations. At operation level changed in the pre and post condition is also identified similarly change in the variable is identified i.e. data type change [32], addition and deletion of the variable or modification in variable. Our proposed approach can be integrated with TESTAF approach which is for the test suite generation from VDM++ specification. TESTAF approach is used for the baseline version test suite generation. In this paper, we have presented case study which shows that our approach provides better results for the regression testing than re-test all strategy [33].

References

[1] Martin H, Horcher M. Improving software tests using Z specifications. In: Proceedings of the 9th international conference of Z users on the Z formal specification notation; 1995. p. 152–66.

[2] Rennard JP. Introduction to genetic algorithms a reference book; 2000. p. 199–203.

[3] Aichernig BK. Automated black-box testing with abstract VDM Oracles. Technical University of Graz, Institute for Software Technology (IST), Miinzgrabenstr.11/II, A-8010 Graz, Austria; 1999. p. 250–9.

[4] Li Y, Wahi NJ. An overview of regression testing ACM SIGSOFT software engineering notes, vol. 24, no. 1; 1999. p. 69-73.

[5] Liang H. Regression testing of classes based on TCOZ specifications. In: Proceedings of the 10th IEEE international conference on engineering of complex computer systems (ICECCS'05), vol. 00; 2005. p. 450–7.

[6] Xu L, Dias M, Richardson D. Generating regression tests via model checking. In: Proceedings of the 28th annual international computer software and applications conference (COMPSAC'04), vol. 01; 2004. p. 336–41.

[7] Beydeda S, Gruhun V. 2002. Class specification implementation graphs and their application in regression testing. Computer software and applications conference, COMPSAC 2002. In: Proceedings. 26th annual international; 2002. p. 835–40.

[8] Singh H, Mirko Sadegh. Test case design based on Z and classification tree method. In: First IEEE international conference on formal engineering methods system technology research, Berlin, Germany; 1997.

[9] Droschl G. Design and application of a test case generator for VDM-SL. Austrian research center seibersdorf and IST-Technical University of Graz.miizgrabenstr.11, 8010 Graz, Austria, Europe; 1999 [no paging].

[10] Nadeem A, Jaffar M. TESTAF: A test automation framework for class testing using object-oriented formal specification, Center for Software Dependability. Pakistan: Muhammad Ali Jinnah University Islamabad; 2005.

[11] Hierons RM, Harman M, Singh H. Automatically generating information from Z specification to support the classification tree method. In: Proceedings of the third international conference of Z and B Users, ZB; 2003 [no paging].

[12] CSK Corporation. VDM++ toolbox user manual revised for V 6.8.1; 2005.

[13] Nadeem A, Jaffar M. TESTAF: a test automation framework for class testing using object-oriented formal specification. J Univ Comput Sci 2005;11(6): 962–85.

[14] Wookcock J, Davies J. Using Z specification, refinement and proof; 1998 [no paging].

[15] Meudec C. Automatic test data generation from VDM-SL specifications. Northern Ireland: Queen's University of Belfast; 1999/2000 [no paging].

[16] Berg MVM, Wigmans Verhoe M. Formal specification of an auctioning system using VDM++ and UML, an industrial usage report. Chess information technology, P.O. Box 5021, 2000 C.A. Haarlen, The Netherlands; 1999 [no paging].

[17] Amayreh A, Zin AM. PROBE: a formal specification-based testing system. In: Proceeding of the 20th international conference on information systems; 1999. p. 400–4.

[18] Agerholm SPJ, Lecoreur, Reichert E. Formal specification and validation at work: a case study using VDM-SL, IFAD Forskerparken 10 DK-5230 Odnese M, Denmark; 1998 [no paging].

[19] Chakrabarti SK, Srikant YN. Specification based regression testing using explicit software engineering advances. In: International conference; October 2006. p. 20.

[20] Chakrabarti SK, Srikant YN. Test sequence computation for regression testing of reactive India Software Engineering Conference. In: Proceedings of the 1st conference on India software engineering conference, Hyderabad, India; 2008. p. 131–2.

[21] Chen CR, Chapman, Chang KH. Test scenario and regression test suite generation from object-z formal specification for object oriented program testing. In: Proceedings of the 37th annual southeast regional conference (CD-ROM), article no. 37; 1999 [no paging].

[22] Burton S, York H. Automated testing from formal specification. Department Of Computer Science, University Of York, YO10 5DD, England; 2000 [no paging].

[23] Chen YRL, Probert, Ural H. 2007. Model-based regression test suite generation using dependence analysis. In: Proceedings of the 3rd international workshop on advances in model-based testing; 2007. p. 54–62.

[24] Fraser G, Aichernig BK, Wotawa F. handling model changes: regression testing and test-suite update with [16] Model-Checkers Institute for Software Technology Graz University of Technology Inffeldgasse 16b/2A-8010 Graz, Austria; 2007.

[25] Foxley E, Salam O. Automatic assessment of Z specification annual joint conference integrating technology in Comp. Science Education; 1997. p. 121–31.

[26] Heimdahl MPE, George D. Test-suite reduction for model based tests: effects on test quality and implications for testing Automated Software Engineering. In: Proceedings of 19th international conference on vols. 20–24; September 2004. p. 176–85.

[27] Horcher HM. Improving software tests using Z specifications. In: Bowen JP, Hinchey MG, editors. ZUM'95: the formal specification notation, vol. 967. Springer-Verlag, Lecture Notes in Computer Science; 1996. p. 152–66.

[28] Horcher HM, Bowen JP, Hinchey MG. Improving software tests using Z specifications. The formal specification notation, vol. 967. Lecture Notes in Computer Science, Springer-Verlag; 1995, p. 152–66.

[29] Rothermel G, Harrold MJ. A framework for evaluating regression test selection techniques. In: Proceedings of the 16th international conference on software engineering; 1994. p. 201–10.

[30] Sergiy AV, Bowen JP. Formalization of software testing criteria using the Z notation computer software and applications conference, (COMPSAC2001) 25th annual international; 2001. p. 351–7.

[31] Stocks PA, Carrington DA. A specification based testing framework. In: The proceedings of the 15th international conference on software eng; 1993 [no paging].

[32] Scullarrd GT. 1998. Test case selection using VDM lecture notes in computer science. In: Proceedings of the Europe symposium on VDM – the way ahead, vol. 328; 1998. p. 178–86.

[33] Marre B, Blanc B. 2005. Test selection strategies for lustre Descriptions in GATeL. Laboratoire Sûreté des Logiciels, CEA/DRT/DTSI/SOL, 91191 Gif sur Yvette CEDEX; 2005 [no paging].