International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 2 Issue 5, May - 2013

# Reducing The Cost And Complexity Of Mutation Testing Using Metadata Versioning

Anu Saini , Raghav Bhasin
*Department of Computer Science,Department of Computer Science,*
*Maharaja Surajmal Institute of Technology,Maharaja Surajmal Institute of Technology,*
*JanakPuri, New Delhi, India.Janakpuri, New Delhi,India.*

Rajat Markan , Rishav Arora
*Department of Computer Science,Department of Computer Science,*
*Maharaja Surajmal Institute of Technology,Maharaja Surajmal Institute of Technology,*
*JanakPuri, New Delhi, India.JanakPuri, New Delhi, India.*

## Abstract

*Mutants which are generated in mutation testing have the same complexity as of the original source code. Thus if the space complexity of the original source code is large, then the cost of mutation testing will also increase due to generation of large number of mutants. In this research paper we have presented the approach to reduce the complexity of mutation testing using metadata versioning. Metadata versioning states that instead of creating a new mutant for the corresponding statement change, we can maintain a copy of original source code and change is incorporated in this copy of source code. Thus we don't need to maintain all copies of mutants, we only need a copy of original source code and a version table that contains the information regarding which statement has to be changed in the copy of source code and what was the previous changed values in that copy. Thus for every mutant, we can change the statements in the copy of original source code and test this against the test cases.*

IJERT

## 1. Introduction

Mutation testing also known as false based testing is one of the important testing techniques that ensure the robustness of test cases. It means test cases should be robust enough to fail the false code also known as mutant code. There are four steps in mutation testing:

Mutants are generated first. Mutants are generated by changing the syntactic elements in the source code. These syntactical changes introduced in the source code are also known as faults.

This is how the mutant is generated as shown in Table 1:

Table I. How the mutant is generated

| Original Source Code | Mutant |
|---|---|
| If(a<b)<br>Print ”mutation testing”<br>Else<br>Print “testing failed” | If(a)<br>Print “mutation testing”<br>Else<br>Print “testing failed” |

Now the test cases are applied to mutants to test the effectiveness of these test cases. The test cases are applied to original source code as well. The test cases should detect faults in the mutant code. Results are computed and then comparison of original source code and mutant code is performed. Mutant is killed if the output of both the original source code and mutant code is same otherwise mutant is kept alive. This was the brief introduction of mutation testing. Now our aim in this study is to reduce the cost of mutation testing. It is obvious that mutants occupy the same space complexity as that of original source code. Thus if we are generating large number of mutants then cost of mutation testing will be very high. To solve this problem, this study uses the concept of metadata versioning. In metadata versioning, if any change is

incorporated in the original files, then that change is reflected in a new table called the metadata version table. Thus all the changed record, time of their change, previous value before that change are kept in that metadata version table. This is the concept that is used in this study to reduce the complexity of mutation testing. What we have is the source code on which the mutation testing is performed and the copy of that source code. Thus instead of creating a new copy of source code for every mutant test, we change statement in the copy of source code using the help of metadata version table. Metadata version table includes all the information related to copy of source code that is information about the, which statement has been changed, what was the previous value before that change, time stamp of change. After the mutant is created we apply the test cases to both the mutant code as well as the original source code. Results are compared and it is decided whether to kill the mutant or it should be kept alive. For creating the second mutant, we don't need to create the second copy of source code, we just need to change few statements in the copy of original source code using the help of metadata version table and then apply the test cases to it. Thus for all the mutants the similar procedure of editing the copy of original source code using metadata version table is performed and effectiveness of source code is tested.

## 2. Proposed Work

Mutants are nothing but the syntactical changes in the one or two statements of original source code. Thus by changing one or two statements in source code a new mutant is generated. But whenever someone creates a new mutant, he creates a new source code and then change the required statements of that source code in order to create a new mutant. Thus each time a new mutant is created, one has to store it separately in order to check the effectiveness of test cases or to perform mutation testing. This creates a storage problem as we have to make many copies of mutants in order to perform the mutation testing. Thus cost of mutation testing increases as the number of mutants increases. This problem of storage can be stored by using the concept of metadata versioning. By using the concept of metadata version table we don't need to store each mutant separately, instead we can have a single copy of original source code and can create mutants from this copy by using the help of metadata version table. Thus space complexity of mutation testing will be reduced and it will be an effective testing technique.

## 3. Implementation of Mutation testing using metadata versioning

This section contains the implementation of our proposed work and how the space complexity can be reduced in mutation testing. There are many metadata versioning techniques. One of the approaches is to use the shadow of tables [1]. But this approach increases complexity. We have used metadata versioning table for our study as proposed by "Metadata Versioning" [2]. Metadata version table tracks the record of any update made to the data base that is if one made update to any row of any table then that changed is stored in metadata version table. That means metadata version table keeps the tracks of changed values, previous values before that change, time stamp of the transaction. We have used this feature of metadata version table. Step by step implementation of mutation testing is explained here. First the program which is under test is given as the input to the system. System accepts this source code and stores it in the database. After the input given to the system the system fetch the source code from the database and creates a copy of that original source code. This copy of source code is also stored in database. Now we want to generate mutants of our source code in order to test the effectiveness of the test cases. Now metadata version table stores the information about mutant that is how mutants are to be generated by altering the statements to original source code. First mutant is generated by changing the statement of copy of source code that is already stored in the database. Thus first mutant is created and stored. Now validation of test case if performed by applying the test cases to both the original source code as well as the first mutant that is stored in the database. The results of both the files are gathered and compared to each other. If the output of both the files is same then that mutant is killed and the copy of original source code is restored by the help of metadata version table because the metadata stores the record of previous values. And if the output of both the files is not the same then that mutant is kept alive and more effective test cases need to consider killing that mutant. For generation of second mutant the same steps are applied that is taking the copy of original source code, changing the syntactical statements for test case generation and then applying the test case to both the mutant and the original source code. Thus a large number of mutants can be created by changing thesmall

IJERT

syntactical statements in the copy of original source code using the help of metadata version table.
Here are a few figures of Implementation of mutation testing:

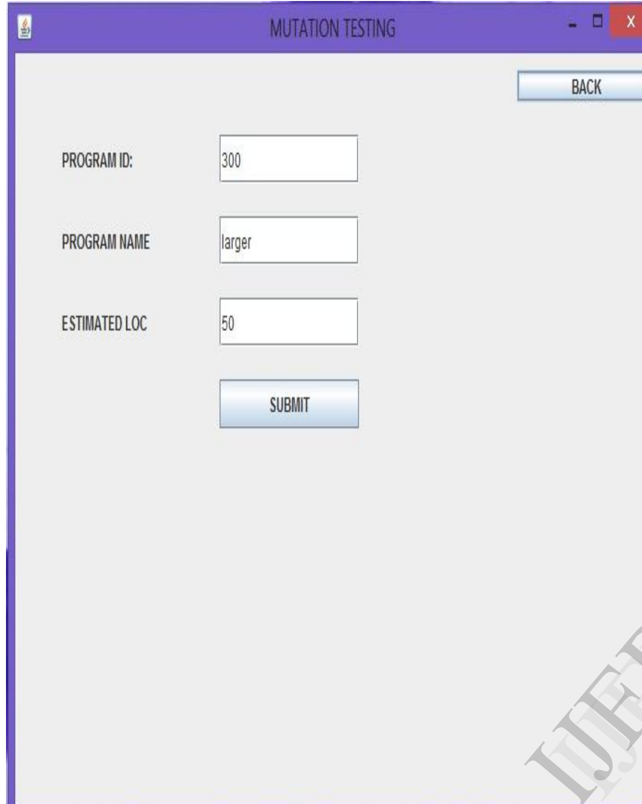


Figure 1. Basic source code information is stored.
Here we can provide basic information about the program such as the program ID, name of the program and the estimated line of codes. This initial data is required in storing the program information. This estimated line of code will create space in the database, so that the program can be stored easily.

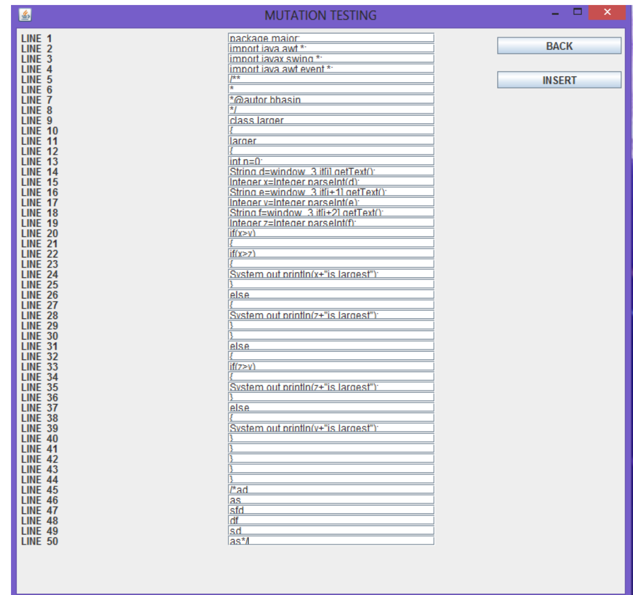


Figure 2. The source code is inserted as input to the system.After the basic information is provided to the system, a new window pops up containing the LOC of source code. The source code is inserted here line by line. After the source code entered to graphical user interface, insert button is pressed and the source code is inserted to the database.

IJERT

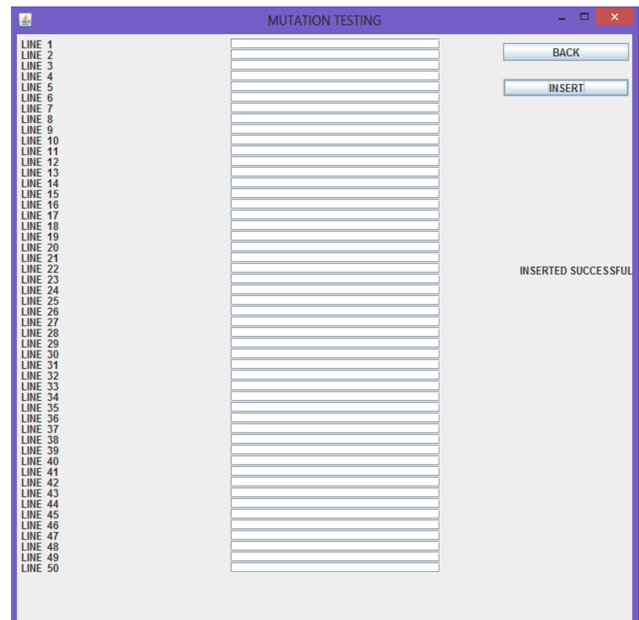


Figure 3: After that the original source code is stored in the database and a copy of source code is generated by the system. When we click on the submit button of figure 2 then the all data from that figure is erase and

ISSN: 2278-0181
Vol. 2 Issue 5, May - 2013

data is stored in the database. When the original source code is stored in the database, then a copy of source code is also generated that performs a major role in reducing the complexity of mutation testing with the help of metadata version table. This is copy is not shown through graphical user interface. Whenever we want to create a new mutant change is implemented in the copy of original source code. And then the test cases are applied to both the original source code and the copy of original source code.

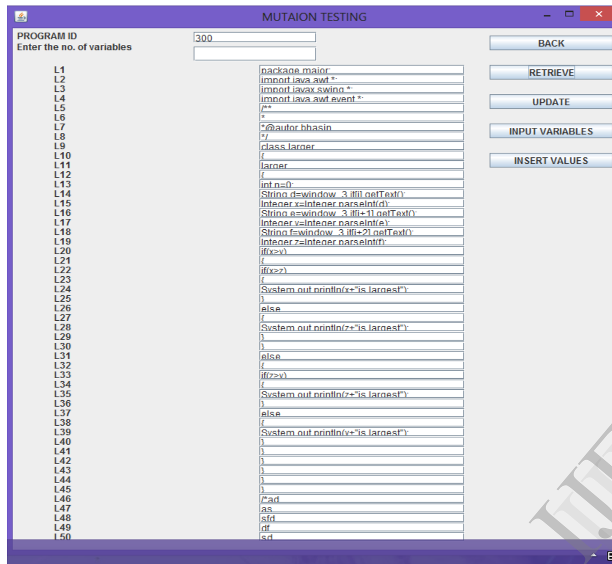


Figure 4. This includes the mutants' generation from the copy of source code. With the help of metadata version table, mutants are generated. Mutants have the same source code except for one or two statements change in the original source code. The statements to be changed can be seen in the metadata version table. And after changing the statements, new mutant is created. If we further want to create a mutant then by using the metadata version table we track the record of previous values as well as the new value to be inserted in the copy of original source code in order to create a new mutant. Thus this is how the mutants are generated one by one with the help of metadata version table and test cases can be applied to both the original source code and mutants in order to test the effectiveness of test cases. Here we can change the value of any mutant by updating the values of mutants, we can also retrieve the mutant from the database.

| VERSION NO | TABLE NAME | PRIMARY KEY | PREVIOUS STATEM... | UPDATED STATEM... | USER | LINE UPDATED | CHANGED TIME |
|---|---|---|---|---|---|---|---|
| 56 | MYPROG1 | 300 | if(x>y) | if(x<y) | SYSTEM | L20 | 06-May-13 20:06 |
| 54 | MYPROG1 | 300 | *@autor bhasin | *@auhtor bhasin | SYSTEM | L7 | 06-May-13 20:06 |
| 55 | MYPROG1 | 300 | int n=0; | int i=0; | SYSTEM | L13 | 06-May-13 20:06 |

Figure 5. This is the metadata version table which records the new value, the previous value and time of transaction. This is the main entity of this study which is required to reduce the complexity of mutation testing. Metadata version table as shown above, records the previous values of mutant, updated values of mutant and information where it is stored in the database. This helps in creation of new mutant from the copy of original source code.

IJERT

## 4. Related Work

Until now very less work is done in reducing the cost of mutation testing. Aditya and W. Eric [3] at Purdue University presented two approaches to reduce the cost of mutation testing. One was, to randomly select x% of mutants. In this approach one may randomly select x% of mutants and can ignore the rest. The value of x can lie between 10 to 100%. The second approach was the constrained mutation criterion [4]. Constrained mutation also takes into account a few mutants and ignores the rest. In the abs constraint mutation we replace every use of x by abs (x), -abs (x), and zpush (x) wherever possible. In ror constraint mutation we replace every relational operator by another relational operator. This study was useful in reducing the cost of mutation testing but the exhaustive testing is not possible in this study and we can have significant loss of very important test cases.

Another study proposed by Macario, Mario and Ignacio [5] reduces the cost of mutation testing by combining the first order mutants to second order mutants. This study uses testooj testing tool for combining first order mutants to second order mutants

which implements three strategies LasttoFirst, DifferentOperator and RandomMix. The numbers of mutants are reduced to half in this study, but again complete exhaustive testing is not possible in this case also.

## 5. Conclusion and Future Scope

This study provides a better approach to reduce the cost of mutation testing using the help of metadata version table. Mutation testing plays a very important role in exhaustive testing, but it does so at the cost of higher space complexity because one needs to maintain multiple copies of mutants. Thus this study provides both the advantages of providing the ultra-high level of mutation coverage and reducing the space complexity of mutation testing. This is very important study and many applications can use this study to test the software system test cases. The test cases of any application can be tested whether these are effective or not. And complete exhaustive testing of any application is possible using this study.

## 6. References

[1]. **Seiderer Peter,** Simple Versioning of Database Entries, **February 6, 2002**
[2]. AnuSaini, RajatMarkan, RishavArora and RaghavBhasin, "Versioning of Metadata", unpublished.
[3]. Aditya P. Mathur and W. Eric Wong, "Reducing the cost of Mutation testing: An Empirical Study."
[4]. A. P. Mathur, "Performance, effectiveness and reliability issues in software testing".
[5]. Macario Polo, Mario Piattini and Ignacio Garcia Rodriguez, "Decreasing the cost of mutation testing with second order mutants".
[6]. http://www.guru99.com/mutation-testing.htm

IJERT

IJERT