International Journal of Modern Engineering Research (IJMER)
www.ijmer.com Vol.3, Issue.2, March-April. 2013 pp-990-995 ISSN: 2249-6645

Implementation of Metamorphic Testing on Spreadsheet Applications

Balinder Singh

Abstract: *End-user programmers do not have broad knowledge about software testing methodologies. Errors are persistent in program due to lack of testing approaches take up by end-user programmers. It is not easy to employ oracle for end-users programs. Metamorphic testing is very effective approach for end-users which also address the oracle problem. In this paper, a metamorphic testing approach for spreadsheet application is used effectively. Metamorphic testing is automated testing approach, which uses expected properties of the objective function to test program without human contribution. This approach defines a set of expected properties called Metamorphic Relations to detect errors in spreadsheet.*

Keywords: *Metamorphic Testing, Oracle, Spreadsheet, Metamorphic Relations.*

I. Introduction

These days, it has become very usual for the end-users to write the program code themselves [4]. End-user programmers are not properly skilled in programming, but they write program to achieve their main goals, such as accounting, office work, scientific work, designing web application etc. Softwares like spreadsheets, databases, web application, simulations etc are being created by end-users. Unluckily, errors are persistent in end-user programs, because End-user programmers do not develop the program according to the software engineering principles as software developed by professional developers. Manuscripts must be in English. These guidelines include complete descriptions of the fonts, spacing, and related information for producing your proceedings manuscripts.

Evidence has pointed out that errors are persistent in Software developed by end-users and may be the reason of losing millions of US dollars [3]. End-user programmers do not have sufficient training for testing, debugging, code inspection etc as like by professional programmers. A variety of software testing methodologies are not easily understood by end-users. So it is required to introduce effective and simple software testing techniques for end- user programmers [6], [11].

Spreadsheet systems have become usual end-user programming environment which is used in different enterprises [5], [12], [18]. Spreadsheets are used for a variety of important tasks such as accounting, scientific computations, data analysis, office work, graphical representation of data.

The Metamorphic Testing (MT) technique that is practical and appropriate where test oracle is very difficult to apply. In software testing, an *oracle* is a procedure that helps the tester to decide whether the output of a program is correct [19]. According to *oracle assumption* [19], Let $foo(x)$ be a function such that $\{foo(x): x \in X\}$ and $pro(x)$ be a program, implementation of $foo(x)$. Let a set of test cases: $T = \{t_1, t_2, \ldots, t_n\} \subset X$, where $n \geq 1$ generated according to some test case selection strategy. Executing the *pro* on these test cases, the tester checks the outputs $pro(t_1), pro(t_2), \ldots, pro(t_n)$ against the expected results $foo(t_1), foo(t_2), \ldots, foo(t_n)$ respectively. If $pro(t_i) = foo(t_i)$, for some i, where $1 \leq i \leq n$, then t_i is known a *successful* test case, otherwise t_i is a *failure-causing* test case. The method by which the tester can decide whether $pro(t_i) = foo(t_i)$ is known as an *oracle* [19]. For example, let $foo(x) = x^2$, the test case t_i will be $\{x = 2.6\}$, the $pro(t_i)$ is = 6.76. The output can be verified by the tester either manually evaluating the multiplication of 2.6 * 2.6 or by using the inverse function to verify whether 6.76 / 2.6 = 2.6. The inverse can be made either using a correct inverse program or manually.

In the literature of software testing, it has to take assumption for the availability of the oracle. In many circumstances, the oracle is not too simple to apply. This is called as the *oracle problem* [19]. For instance, the output of complicated computations program, such as solving partial differential equations; when compilers are tested, it is very difficult to test the equivalence between source code and the generated object code; conducting testing for a program that compute combinatorial problems, testing program which performs simulations, drawing the complicated graphics to the monitor etc. Indeed, when the oracle is available and if it is not automated, then human tester performs comparison and predication on the test result manually, that means it becomes too expensive, time consuming and prone to error [2], [21]. As a matter of fact, in software testing, the oracle problem has been one of the most difficult tasks [2].

A Metamorphic Testing method [19] was introduced by *Chen et al.* to use successful test cases and to address the oracle problem. The idea behind this method is to generate follow-up test cases based on the existing successful test cases. The output of the follow-up test cases can be checked by using relation (called metamorphic relations (MR)), these are the relations between multiple executions of the target program. Follow-up test cases can be automatically generated, executed, and the verification of program *pro* does not require an oracle.

In this paper, we present the implementation of metamorphic testing for spreadsheet applications. We develop six metamorphic relations to find out errors in spreadsheet application. The rest of the paper is organized as follows. Section II introduces Background details of Metamorphic testing and Spreadsheet. Problem Analysis has been done in section III, in section IV Proposed Work and Conclusion has been done in section V.

II. Background Details

2.1. Metamorphic Testing

Metamorphic testing (MT) technique is a simple and cost-effective approach to find out the important information from the existing successful test cases (test cases those display no failure) to generate the follow-up test cases [7]. MT can be applied with the combination of other test case selection strategies which generate the initial test cases. Let *foo*(*x*) be a function such that {*foo*(*x*): $x \in X$} and *pro*(*x*) be a program, implementation of *foo*(*x*). To test program *pro*, the tester has adopted *Ss* as the test case selection strategy, such as branch coverage testing, data flow testing or random testing [1]. Let a test set $T = \{t_1, t_2, ..., t_n\} \subset X$, where $n \geq 1$, generated according to *Ss*. Execute the *pro* on *T* produces the output results *pro*(t_1), *pro*(t_2),...., *pro*(t_n). If the output results *pro*(t_1), *pro*(t_2),...., *pro*(t_n) notice any failure, testing is stopped and the program is started to be debugged; otherwise *T* is known as a set of *successful* test cases.

MT can be performed to make use of *T*, where in other testing methods, successful test cases have been ignored and considered useless [9]. Based on the successful test set *T*, follow-up test cases $T'=\{t'_1, t'_2, ..., t'_n\} \subset X$ can be automatically generated by MT. Further, program *pro* is executed against some necessary expected properties (called metamorphic relations) of the target function *foo*. MR contains multiple executions of the target program. As an example, If $foo(x) = \beta^y$, where β is rational number excluding zero, *y* is an integer, then a program *pro* implemented on function *foo*(*x*) which calculates the result of target function. The property $\beta^y * \beta^{-y} = 1$ is considered as a metamorphic relation. Metamorphic testing checks the program against some necessary excepted properties called Metamorphic Relations. Therefore, Metamorphic testing executes the implemented program twice, firstly, for a successful test case, i.e. $t_i = \{\beta = -3.68, y = -6\}$, secondly, for follow-up test case generated by MT, i.e. t'_i: $\{\beta = -3.68, y = 6\}$. Finally, the computed results of both executions are checked against the expected property $pro(t_i) * pro(t'_i) = 1$. If this property is not true, then a fault is detected. If the given property is true, then MT does not specify that program's output may be correct or not. In other words, it is necessary for MT to check the properties of target function, which may not be sufficient for correctness of implemented program. Limitation of this type is found in all software testing methods.

The successfully use of MT in many areas such as numerical, spreadsheet and database, graph theory, computer graphics, compilers, interactive software were described in [13], [14]. Guidelines for selecting good metamorphic relations in metamorphic testing have been proposed in [10], [16]. To alleviate the oracle problem, fault based testing has been described in [8].

2.2. Spreadsheet Basis

A spreadsheet is a computing application that displays a rectangular table (or grid) of information, consisting of text and numbers, where values sit in cells. Spreadsheets allow defining the type of data for each cell (usually limited to texts and numbers) and defining how different cells depend on each other ([15]). Formulas describe relationships between cells. As earlier mentioned spreadsheet is end-users application and are very errors-prone. Estimates point out that half of all the spreadsheets which are created contain errors, this number may be more than as 90% [20].

In [17], a hierarchical classification of spreadsheet errors has been developed which is based on the nature as well as characteristics of the error and development life cycle of spreadsheet. In [12], three categories of spreadsheet errors namely, mechanical, logic and omission errors are described. *Mechanical* errors happen due to mistyping or pointing to a wrong cell. *Logic* errors occur due to selecting the wrong algorithm for a particular formula cell or incorrectly use of formula, resulting incorrect output. *Omission* errors occur when a spreadsheet developer does not have complete knowledge of the problem and resulting, an incomplete spreadsheet model of the problem. In experiment, 1.8% of spreadsheet cells contain mechanical errors and logic errors of 4.1% [3].

In [13], concepts, procedures and applications are defined for Metamorphic Testing but not sufficient gain for spreadsheet Applications. In [14], Metamorphic testing framework is introduced for finding error in spreadsheet using circular shift columns. If more than one column involved in formula have same value in their cells, then circular shifting column is not sufficient to detect errors. We have developed MRs that can detect errors for this problem also.

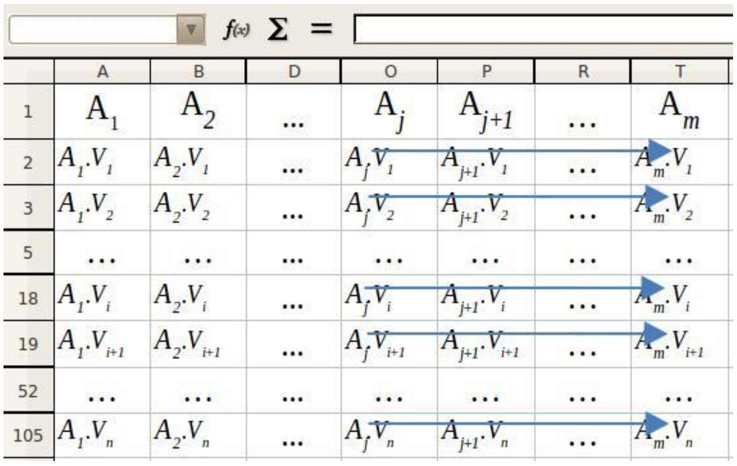
III. Problem Analysys

1.1. Problem

A spreadsheet may have many rows and columns. It is not easy to check huge amount of rows and columns manually as it is very time consuming. Test oracle [19] is too difficult for spreadsheet, if it is available and cannot be automated, resulting too expensive as verification of spreadsheet having huge amount to data is very time consuming. Therefore, to check spreadsheet application, an effective testing approach is required (generated**)** that should be simple to implement and less time consuming. Following assumptions are made:

3.1.1 It is very difficult to create test oracle on the huge amount of data in spreadsheet. Metamorphic testing is carried out to remove this problem. Many Metamorphic relations are generated that are based on the properties of target spreadsheet. Metamorphic testing does not guarantee that *program under test* produces correct output. It guarantees only to check the properties of target function.

3.1.2. A typical spreadsheet may have hundreds of rows and columns. Checking the huge amount of rows and columns manually is very time consuming, although, Metamorphic testing method [14] is applied to check errors in huge amount of data for spreadsheet quickly. As an illustrative instance, lets us take a simplified **Excel** spreadsheet shown below in Fig.1.

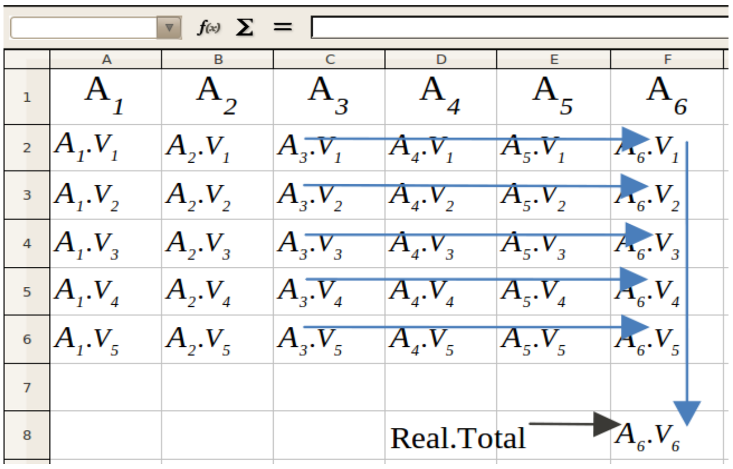


	A	B	D	O	P	R	T
1	A_1	A_2	...	A_j	A_{j+1}	...	A_m
2	$A_1.V_1$	$A_2.V_1$	...	$A_j.V_1$	$A_{j+1}.V_1$	...	$A_m.V_1$
3	$A_1.V_2$	$A_2.V_2$	...	$A_j.V_2$	$A_{j+1}.V_2$	...	$A_m.V_2$
5	...	...	...	...	...	...	...
18	$A_1.V_i$	$A_2.V_i$	...	$A_j.V_i$	$A_{j+1}.V_i$	...	$A_m.V_i$
19	$A_1.V_{i+1}$	$A_2.V_{i+1}$	...	$A_j.V_{i+1}$	$A_{j+1}.V_{i+1}$	...	$A_m.V_{i+1}$
52	...	...	...	...	...	...	...
105	$A_1.V_n$	$A_2.V_n$	...	$A_j.V_n$	$A_{j+1}.V_n$	...	$A_m.V_n$

Fig.1: Spreadsheet

In Fig.1, A_j denotes the attribute, where $1 \le j \le m$ and $A_j.V_i$ denotes the value of that attribute, where $1 \le i \le n$. Values of attributes may be numeric and character type. Horizontal arrows in Fig.1, indicate the columns (A_j, $A_j + 1$,..., A_{m-1}) are involving in formula, which is laying in column A_m. Formula may be of finding sum, average, percentage, calculating discount or tax etc. on the attributes values which are involved in formula. The spreadsheet representation of this type can be used for many real life applications such as calculating total price of sold/purchased items, calculating the discount or tax on items, finding average marks or aggregate marks of different class tests in a teaching institutes etc. There may be error in any cells which contain formula [14]. Error of this type can occur when someone tries to enter the formula manually without copying formula from adjacent cells. The error cannot be noticed due to huge amount of data in spreadsheet.

This may be very harmful, if the cell containing wrong formula is dragged to copy the formula for cells below it. Formula errors are very difficult to detect as mentioned in [3]. On the other hand, by using MT, problems of this type can be easily handled. Take the example of spreadsheet mentioned above to apply MT approach. Suppose Fig.2 describes a spreadsheet as *Real Spreadsheet*, which contains few records so that MT approach can be shown easily.



	A	B	C	D	E	F
1	A_1	A_2	A_3	A_4	A_5	A_6
2	$A_1.V_1$	$A_2.V_1$	$A_3.V_1$	$A_4.V_1$	$A_5.V_1$	$A_6.V_1$
3	$A_1.V_2$	$A_2.V_2$	$A_3.V_2$	$A_4.V_2$	$A_5.V_2$	$A_6.V_2$
4	$A_1.V_3$	$A_2.V_3$	$A_3.V_3$	$A_4.V_3$	$A_5.V_3$	$A_6.V_3$
5	$A_1.V_4$	$A_2.V_4$	$A_3.V_4$	$A_4.V_4$	$A_5.V_4$	$A_6.V_4$
6	$A_1.V_5$	$A_2.V_5$	$A_3.V_5$	$A_4.V_5$	$A_5.V_5$	$A_6.V_5$
7						
8				Real.Total		$A_6.V_6$

Fig.2: Real spreadsheet

In Fig.2, A_6 contains a formula, which may be according to the user requirement. For example, calculating the discount on attributes, therefore, the formula is "$A_6 = A_3 -(A_3 *40\%) + A_4 -(A_4 *50\%) + A_5 -(A_5 *60\%)$". Cell F_8 describes the *Total* of all the values of A_6, calculated from the formula. Suppose the formula in the cell F_5 has been mistakenly entered as "$A_6 = A_3 -(A_3 *60\%) + A_4 -(A_4 *50\%) + A_5 -(A_5 *40\%)$". Error of this type cannot be identified easily, if the $A_3.V_4 = A_5.V_4$, then $A_6.V_4$ is correct according to real formula, although, the formula in cell F_5 is really wrong. Errors of this type can be easily tackled by identifying suitable metamorphic relations corresponding to the nature of spreadsheet.

1.2. Metamorphic Testing

For the problem described above, oracle is not effective to apply because (1) spreadsheet contain huge amount of data, (2) spreadsheet allows performing complicated function on numeric values. In this situation, if oracle is not automated then it is too time consuming, expensive and prone to error. For this problem, Metamorphic Testing is effective and less time consuming, resulting not too expensive. Other advantage of MT is that it only requires the domain knowledge of spreadsheet to generate the metamorphic relations. On the other hand, spreadsheet is generally used by end-users and end-users contain great knowledge about their application domain. But they have no knowledge about software testing approach;

therefore, metamorphic testing for spreadsheet applications can be very efficient for end-users where test oracle is not available. A spreadsheet has many features like copy, paste, sorting (ascending, descending) etc and many functions. i.e. sum, average, count, etc which can be used to generate the metamorphic relations. All Metamorphic relations are generated manually using spreadsheet features and functions.

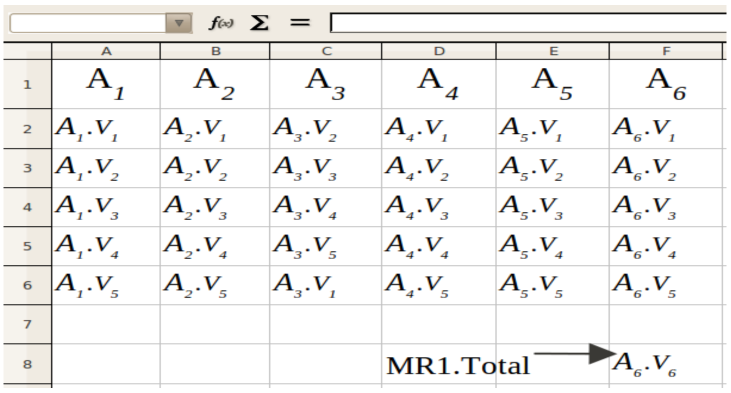
IV. Our Approach

4.1. Metamorphic Relation generation

For each and every Metamorphic Relation, we have to use *Real Spreadsheet* to create a new spreadsheet by *copy* and *paste*. The operations used in Metamorphic Relations are easily applied by "copy and paste" on the attributes which are involved in formula. We have generated six Metamorphic Relations as follow:

1.2.1. Metamorphic Relation1

First, create a new spreadsheet as MR1spreadsheet as shown in Fig.3 by copying the *Real Spreadsheet*. Apply an operation (on MR1spreadsheet) i.e. "circular shift up" [14] on the values of attributes. The *operation* which shift the values of a single attribute circularly one place up at a time Then compare the *Total* of all the values of A_6, calculated by the formula, from *Real Spreadsheet* and MR1spreadsheet i.e. *Real. Total = MR1.Total* as shown in Fig.2 and Fig.3 in cells F_8. If both *Totals* are not equal, it means *Real Spreadsheet* has an error; otherwise perform operation "circular shift up" again on same attribute and then check the equivalence between *Totals*. This process is repeated *n*-1 time on single attribute, where *n* is number of values of that attribute. This process is carried on until error is detected or all attributes (involved in formula) are not passed out by this process.



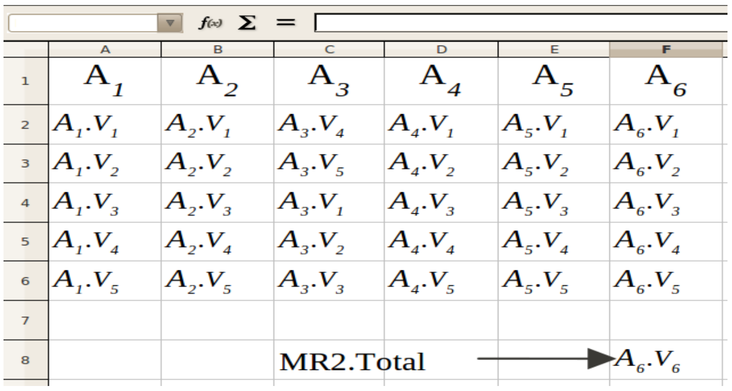
	A	B	C	D	E	F
1	A_1	A_2	A_3	A_4	A_5	A_6
2	$A_1.V_1$	$A_2.V_1$	$A_3.V_2$	$A_4.V_1$	$A_5.V_1$	$A_6.V_1$
3	$A_1.V_2$	$A_2.V_2$	$A_3.V_3$	$A_4.V_2$	$A_5.V_2$	$A_6.V_2$
4	$A_1.V_3$	$A_2.V_3$	$A_3.V_4$	$A_4.V_3$	$A_5.V_3$	$A_6.V_3$
5	$A_1.V_4$	$A_2.V_4$	$A_3.V_5$	$A_4.V_4$	$A_5.V_4$	$A_6.V_4$
6	$A_1.V_5$	$A_2.V_5$	$A_3.V_1$	$A_4.V_5$	$A_5.V_5$	$A_6.V_5$
7						
8				MR1.Total $\longrightarrow$		$A_6.V_6$

Fig.3: MR1spreadsheet

Shifting is easily done manually by using *copy* and *paste* the values of attribute. Fig.3 shows the *operation* on *A*3 only one time.

1.2.2. Metamorphic Relation2

Like Metamorphic Relation1, take copy of *Real Spreadsheet* and paste to make a new spreadsheet as MR2spreadsheet, shown in Fig.4.



	A	B	C	D	E	F
1	A_1	A_2	A_3	A_4	A_5	A_6
2	$A_1.V_1$	$A_2.V_1$	$A_3.V_4$	$A_4.V_1$	$A_5.V_1$	$A_6.V_1$
3	$A_1.V_2$	$A_2.V_2$	$A_3.V_5$	$A_4.V_2$	$A_5.V_2$	$A_6.V_2$
4	$A_1.V_3$	$A_2.V_3$	$A_3.V_1$	$A_4.V_3$	$A_5.V_3$	$A_6.V_3$
5	$A_1.V_4$	$A_2.V_4$	$A_3.V_2$	$A_4.V_4$	$A_5.V_4$	$A_6.V_4$
6	$A_1.V_5$	$A_2.V_5$	$A_3.V_3$	$A_4.V_5$	$A_5.V_5$	$A_6.V_5$
7						
8			MR2.Total $\longrightarrow$			$A_6.V_6$

Fig.4: MR2spreadsheet

In MR2spreadhsheet, apply an operation "circular shift down" [14] which performs shifting the values of an attribute circularly only one place down at a time. Similar to MR1, compare the *Total* of *Real Spreadsheet* and MR2spreadsheet. If both *Totals* (*Real. Total = MR2.Total*) are not equal, it means *Real Spreadsheet* has an error; otherwise perform operation "circular shift down" again on same attribute and then check the equivalence. This process is repeated *n*-1 time on single attribute, where *n* is declared earlier. We apply this operation until error is detected or all the attributes involved in formula are not passed out by this process. *Operation* is performed manually by *copy* and *paste*. In Fig.4, this *operation* is performed only two times on A_3, manually by “copy and paste”.

1.2.3. Metamorphic Relation3

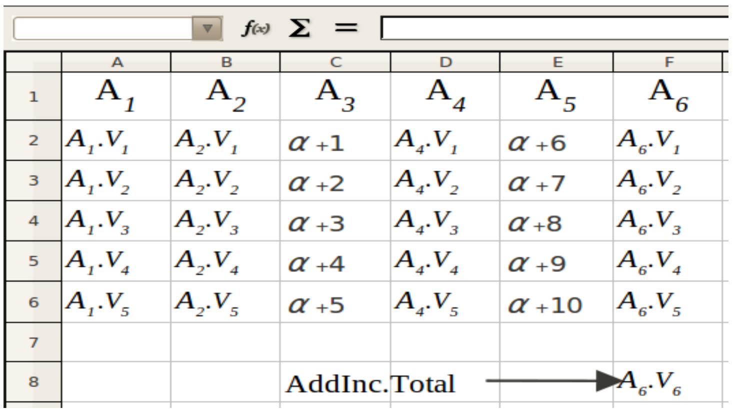
Sorting is one of the main features of spreadsheet. Create a new spreadsheet as MR3spreadsheet by copying *Real Spreadsheet*. MR3spreadsheet is used to perform an operation “ascending sort” which sort the values of an attribute in ascending order. Then, compare the *Total* of both the spreadsheets i.e. *Real Spreadsheet* and MR3spreadsheet. If there is no equivalence between both *Totals*, it means error in *Real Spreadsheet*; otherwise perform the *operation* on the second attribute and check the equivalence between *Totals* and so on until error is detected or this procedure is not carried on for all the attributes involved in formula.

1.2.4. Metamorphic Relation4

Like Metamorphic Relation3, first, form a new spreadsheet as MR4spreadsheet by copying the *Real Spreadsheet*. In MR4spreadsheet, apply “descending sort” as an operation on the values of an attribute. The operation produces result sorting the values in descending order. Then compare the *Total* of both spreadsheets i.e. *Real Spreadsheet* and MR4spreadsheet. If there is no equivalence between both *Totals*, it means, there is an error in *Real Spreadsheet*; otherwise perform the *operation* on second attribute and check the equivalence between *Totals* and so on until error is detected or this process is not applied for all the attributes involved in formula.

1.2.5. Metamorphic Relation5

If in *Real Spreadsheet*, more than one attribute involved in formula contain same values i.e. $A_j.V_i = \alpha$, Then Metamorphic Relation5 can be very effective to detect error. First, create a new spreadsheet as *Additive Increase* (as shown in Fig.5) by copying the *Real Spreadsheet*. In *Additive Increase*, add consecutive integer with value α, for all the attributes which have same values and also involved in formula. Suppose, A_3 and A_5 have same values in *Real Spreadsheet*, then addition of consecutive integer has been also shown in *Additive Increase* in Fig.5.



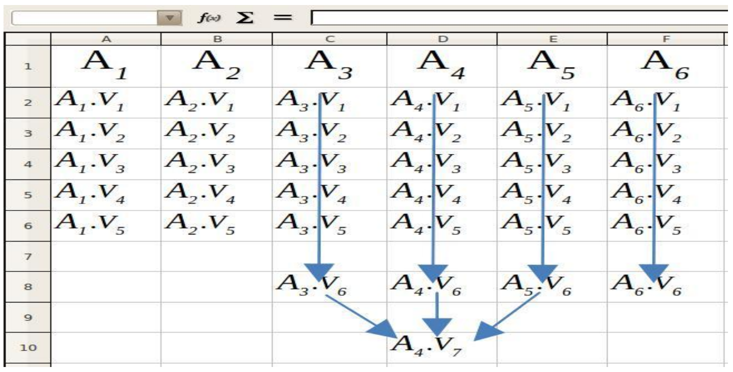
	A	B	C	D	E	F
1	A_1	A_2	A_3	A_4	A_5	A_6
2	$A_1.V_1$	$A_2.V_1$	$\alpha+1$	$A_4.V_1$	$\alpha+6$	$A_6.V_1$
3	$A_1.V_2$	$A_2.V_2$	$\alpha+2$	$A_4.V_2$	$\alpha+7$	$A_6.V_2$
4	$A_1.V_3$	$A_2.V_3$	$\alpha+3$	$A_4.V_3$	$\alpha+8$	$A_6.V_3$
5	$A_1.V_4$	$A_2.V_4$	$\alpha+4$	$A_4.V_4$	$\alpha+9$	$A_6.V_4$
6	$A_1.V_5$	$A_2.V_5$	$\alpha+5$	$A_4.V_5$	$\alpha+10$	$A_6.V_5$
7						
8			AddInc.Total		$\longrightarrow$	$A_6.V_6$

Fig.5: Additive Increase

Finally, *Additive Increase* is copied and paste to make a new spreadsheet as MR5spreadsheet. Then apply any one of Metamorphic Relations like (Metamorphic Relation1, Metamorphic Relation2, Metamorphic Relation3, and Metamorphic Relation4) on the MR5spreadsheet. Finally, compare the *Totals* (*AddInc. Total* = *MR5.Total*) of *Additive Increase* and MR5spreadsheet to check the error.

1.2.6. Metamorphic Relation6

Create a new spreadsheet as MR6spreadsheet by copying the *Additive Increase*.



	A	B	C	D	E	F
1	A_1	A_2	A_3	A_4	A_5	A_6
2	$A_1.V_1$	$A_2.V_1$	$A_3.V_1$	$A_4.V_1$	$A_5.V_1$	$A_6.V_1$
3	$A_1.V_2$	$A_2.V_2$	$A_3.V_2$	$A_4.V_2$	$A_5.V_2$	$A_6.V_2$
4	$A_1.V_3$	$A_2.V_3$	$A_3.V_3$	$A_4.V_3$	$A_5.V_3$	$A_6.V_3$
5	$A_1.V_4$	$A_2.V_4$	$A_3.V_4$	$A_4.V_4$	$A_5.V_4$	$A_6.V_4$
6	$A_1.V_5$	$A_2.V_5$	$A_3.V_5$	$A_4.V_5$	$A_5.V_5$	$A_6.V_5$
7						
8			$A_3.V_6$	$A_4.V_6$	$A_5.V_6$	$A_6.V_6$
9						
10				$A_4.V_7$		

Fig.6: MR6spreadsheet

In MR6spreadsheet as shown in Fig.6, calculate the total of all the values of A_3, A_4, and A_5 which are involved in formula, in the cells C8, D8 and E8 respectively. Apply *real* formula on the values of the cells C8, D8 and E8. Calculate the result in cell D10. If the $A_4.V_7 \neq A_6.V_6$, then *Real Spreadsheet* has error; otherwise it has no error.

V. Conclusion

In this paper, metamorphic testing is employed for spreadsheet due to lack of test oracle. Test oracle for spreadsheet is very difficult to use, as a spreadsheet contains huge amount of data. Apply oracle for spreadsheet containing large range of data is very time consuming, this may result of being costly. Therefore, we use MT for spreadsheet, which is very simple and easy to implement. It also addresses the oracle problem by generating the metamorphic relations according to the expected properties of the target function. MT is very effective for end-users because end-user have domain knowledge of spreadsheet that helps to generate good metamorphic relations. Finally, we concluded that MT can be used to check the spreadsheet effectively and quickly.

References

[1] B. Beizer. Software Testing Techniques. Van Nostrand Reinhold, New York, 1990.
[2] Manolache, L. I. and Kourie, D. G. Software testing using model programs, Software: Practice and Experience, 31 (13), 2001, pp. 1211-1236.
[3] R. Panko. Finding spreadsheet errors. InformationWeek, Issue 529, page 100, May 1995.
[4] B. Boehm, C. Abts, A. Brown, S. Chulani, B. Clark, E. Horowitz, R. Madachy, D. Reifer, and B. Steece. Software Cost Estimation with Cocomo II. Prentice Hall PTR, Upper Saddle River, NJ, 2000.
[5] Ballinger, D., Biddle, R., and Noble, J. Spreadsheet Structure Inspection Using Low Level Access and Visualization. In Proceedings of the Fourth Australasian Conference on User Interfaces (Darlinghurst, Australia, 2003), Australian Computer Society, Inc., pp. 91-94.
[6] G. Rothermel, M. Burnett, L. Li, C. Dupuis, and A. Sheretov. A methodology for testing spreadsheets. ACM Transactions on Software Engineering and Methodology, 10(1):110-147, 2001.
[7] T.Y. Chen, S.C. Cheung, and S.M. Yiu. Metamorphic testing: a new approach for generating next test cases. Technical Report HKUST-CS98-01, University of Science and Technology, Hong Kong, 1998.
[8] Chen, T. Y., Tse, T. H., Zhou, Z. Q.: Fault-based testing without the need of oracles. Information and Software Technology 45 (2003), pp. 1-9.
[9] G. J. Myers. The Art of Software Testing. Wiley, New York, 1979.
[10] J. Mayer, and R. Guderlei, "An empirical study on the selection of good metamorphic relations", In Proc. of the 30th Annual International Computer Software and Applications Conference (COMPSAC), 2006, pp. 475-484.
[11] M. Burnett, C. Cook, and G. Rothermel. End-user software engineering. Communications of the ACM, 47(9), 2004, pp. 53-58.
[12] Panko, R. R. Spreadsheet Errors: What We Know. What We Think We Can Do. In Proceedings of the European Spreadsheet Risk Interest Group Symposium (2000).
[13] Z. Q. Zhou, D. H. Huang, T. H. Tse, Z. Yang, H. Huang, and T. Y. Chen. Metamorphic testing and its applications. In Proceedings of the 8th International Symposium on Future Soft- ware Technology (ISFST 2004), Xian, China, October 20-22 2004.
[14] Chen TY, Kuo FC, Zhou ZQ. An Effective Testing Method for End-User Programmer. In Proceedings of the First Workshop on End-User Software Engineering. 2005. pp. 1-5.
[15] Obrenović, Gašević, D., "End-User Service Computing: Spreadsheets as a Service Coordination Interface," IEEE Transactions on Services Computing, Vol. 1, No. 4, 2008, pp. 229-242.
[16] T. Y. Chen, D. H. Huang, T. H. Tse, and Z. Q. Zhou. Case studies on the selection of useful relations in metamorphic testing. In Proceedings of the 4th Ibero-American Symposium on Software Engineering and Knowledge Engineering (JIISIC 2004), Polytechnic University of Madrid, Madrid, Spain, 2004, pp. 569-583.
[17] Rajalingham, K., Chadwick. D., and Knight, B. (2000). "Classification of Spreadsheet Errors." Proceedings of the European Spreadsheet Risks Interest Group Annual Conference, Greenwich, UK. pp. 23-34.
[18] Wilson, A., Burnett, M., Beckwith, L., Granatir, O., Casburn, L., Cook, C., Durham, M., and Rothermel, G. Harnessing Curiosity to Increase Correctness in End-User Programming. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (New York, NY, USA, 2003), ACM, pp. 305-312.
[19] E.J. Weyuker. On testing non-testable programs. The Computer Journal, 25(4), 1982, pp. 465-470.
[20] "What we know about spreadsheet errors, Human Error Website, Honolulu, HI: University of Hawai'i", Retrieved May 15, 2005 http site, "http://panko.cba.hawaii.edu/ HumanErr/"
[21] Hamlet, D. Predicting dependability by testing, In Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 1996), ACM Press, New York, 1996, pp. 84-91.