An Approach to Cost Effective Regression Testing in Black-Box Testing Environment

Prof. A. Ananda Rao [1] and Kiran Kumar J [2]

[1] Prof. of CSE and Principal, JNTUA
Anantapur, Andhra Pradesh, India

[2] Software Engineer, IBM
India

Abstract

Regression testing is an expensive and frequently executed maintenance activity used to revalidate the modified software. As the regression testing is a frequently executed activity in the software maintenance phase, it occupies a large portion of the software maintenance budget. Any reduction in the cost of regression testing would help to reduce the software maintenance cost. The current research is focused on finding the ways to reduce the regression testing cost. In this paper, an approach to test suite reduction for regression testing in black box environment has been proposed. This type of approach has not been used earlier. The reduced regression test suite has the same bug finding capability and covers the same functionality as the original regression test suite. The proposed approach is applied on four real-time case studies. It is found that the reduction in cost of regression testing for each regression testing cycle is ranging between 19.35 and 32.10 percent. Since regression testing is done more frequently in software maintenance phase, the overall software maintenance cost can be reduced considerably by applying the proposed approach.

Keywords: *Software maintenance cost, ETL DB Component, reduced test suite, reduced regression test suite, test case design, regression testing cost reduction.*

1. Introduction

The estimated cost of software maintenance activities occupies as much as two-thirds of the total cost of software production [18]. Regression testing is a critical part of the software maintenance that is performed on the modified software to ensure that the modifications do not adversely affect the unchanged portion of the software. As regression testing is performed frequently in software maintenance, it accounts for a large portion of the maintenance costs [9, 10, 11]. Regression testing is "selective retesting of a system or component to verify that modifications have not caused unintended effects and that the system or component still complies with its specified requirements." [1].

Numerous techniques have been proposed to deal with the regression testing costs. Regression test selection techniques select a subset of existing test case set for execution, depending on criteria such as changes made to the software. Test suite reduction techniques reduce the test suite permanently by identifying and removing redundant tests. Test case prioritization techniques retain the complete test suite, but change the order of test cases prior to execution, attempting to find the defects earlier during the testing. During software maintenance phase, testing teams need to run regression test case set on many intermediate builds, to ensure that the bug fixes or enhancements made to the software do not adversely affect unchanged portions of the software. In this paper, an approach to reduce the total number of regression test cases in black box environment without affecting the defect coverage and functionality coverage of software is proposed. This reduction in the regression test suite size will reduce the effort and time required by the testing teams to execute the regression test suite.

Most of the existing approaches consider test suite which contain, test cases to test the functionality, boundary values, stress, and performance of the software. Any reduction in this test suite size will reduce the testing time, effort, and cost. Many of the test cases in this test suite belong to the functionality and boundary values of the software. The proposed approach is applied on the original test suite to derive the reduced test suite. This reduced test suite covers the same functionality of the software as the original test suite. A regression test selection method is applied on this reduced test suite, to get the reduced regression test suite. This reduced regression test suite covers the same defect coverage and functionality as the

original regression test suite. In this proposed approach, it is shown that the two aspects of testing, that is testing for functionality and testing for boundary values can be tested with reduced test suite as these two aspects can be tested together simultaneously in most of the situations. The situations where these two aspects can be tested simultaneously, is also shown with help of the case-studies. In this paper, testing simultaneously means, a single test case can cover both the above mentioned aspects for a particular situation. The proposed approach is applied on four real-time case studies and also estimated the reduction in cost of regression testing using a cost estimation model. It is found that the reduction in cost per one regression testing cycle is ranging between 19.35 and 32.10 percent. Since regression testing is more frequently done activity in software maintenance phase, the overall regression testing cost can be reduced considerably by applying the proposed approach.

The rest of the paper is organized as follows: Section II reviews the various regression testing techniques and summarizes related work. Section III describes the proposed approach to cost effective regression testing for black-box testing environment. Section IV describes the Empirical studies and results of the proposed approach. Section V concludes and discusses future work.

2. Related Work

Researchers, practitioners and academicians proposed various techniques on test suite reduction, test case prioritization, and regression test selection for improving the cost effectiveness of the regression testing.

Rothermel and Harrold presented a technique for regression test selection. Their algorithms construct control flow graphs for a procedure or program and its modified version and use these graphs to select tests that execute changed code from the original test suite [9]. James A. Jones and Mary Jean Harrold proposed new algorithms for test suite reduction and prioritization [2]. Saifur-Rehman Khan, Aamer Nadeem proposed a novel test case reduction technique called TestFilter that uses the statement-coverage criterion for reduction of test cases [3]. T. Y. Chen and M. F. Lau presented dividing strategies for the optimization of of a test suite [4]. M. J. Harrold etal presented a technique to select a representative set of test cases from a test suite that provides the same coverage as the entire test suite [5]. This selection is performed by identifying, and then eliminating, the redundant and obsolete test cases in the test suite. This technique is illustrated using data flow testing methodology. A recent study by Wong, Horgan, London, and Mathur [6], examines the costs and benefits of test suite minimization. Rothermel et al [7] described several techniques for using test execution information to prioritize test cases for regression testing, including: techniques that order test cases based on their total coverage of code components, techniques that order test cases based on their coverage of code components not previously covered, and techniques that order test cases based on their estimated ability to reveal faults in the code components that they cover.

Most of the techniques described in the above papers assume that source code of the software is available to the testing engineer at the time of testing. But in most of the organizations the testing is done in black box environment and the source code of the software is not available to the testing engineers. In this paper, an approach to reduce cost of software regression testing in black box environment, without affecting the functionality coverage, is presented.

3. The Proposed Approach

The estimated cost of software maintenance exceeds 70% of total software costs [16], and large portion of this maintenance expense is devoted to regression testing. Regression testing is a frequently executed activity, so reducing the cost of regression testing would help in reducing cost of the software maintenance.

The proposed approach is shown in three phases (Fig.1). In Phase 1 (Fig. 1), the “Reduced Test Suite” is derived by applying the proposed approach on the Original test suite. Phase 1 of the approach is already proposed by the authors in [17], and in Phase 2 (Fig. 1), the “Reduced Regression Test Suite” is derived by applying a regression test selection method on the “Reduced Test Suite” that is derived in the Phase 1. In Phase 3, a testing cost-estimation model is applied on the reduced regression test suite and empirically calculated the regression testing cost reduction by the proposed approach.

Phase 1: Deriving the “Reduced Test Suite”

A large number of test cases are derived by applying various testing techniques to test complete functionality of a software product. This test suite contains test cases to test functionality, boundary values, stress, and performance of the software product. Majority of these test cases will be test cases that test the functionality and boundary values. The Phase 1 of the proposed approach is focused on reducing test cases considering test cases that test functionality and boundary values.

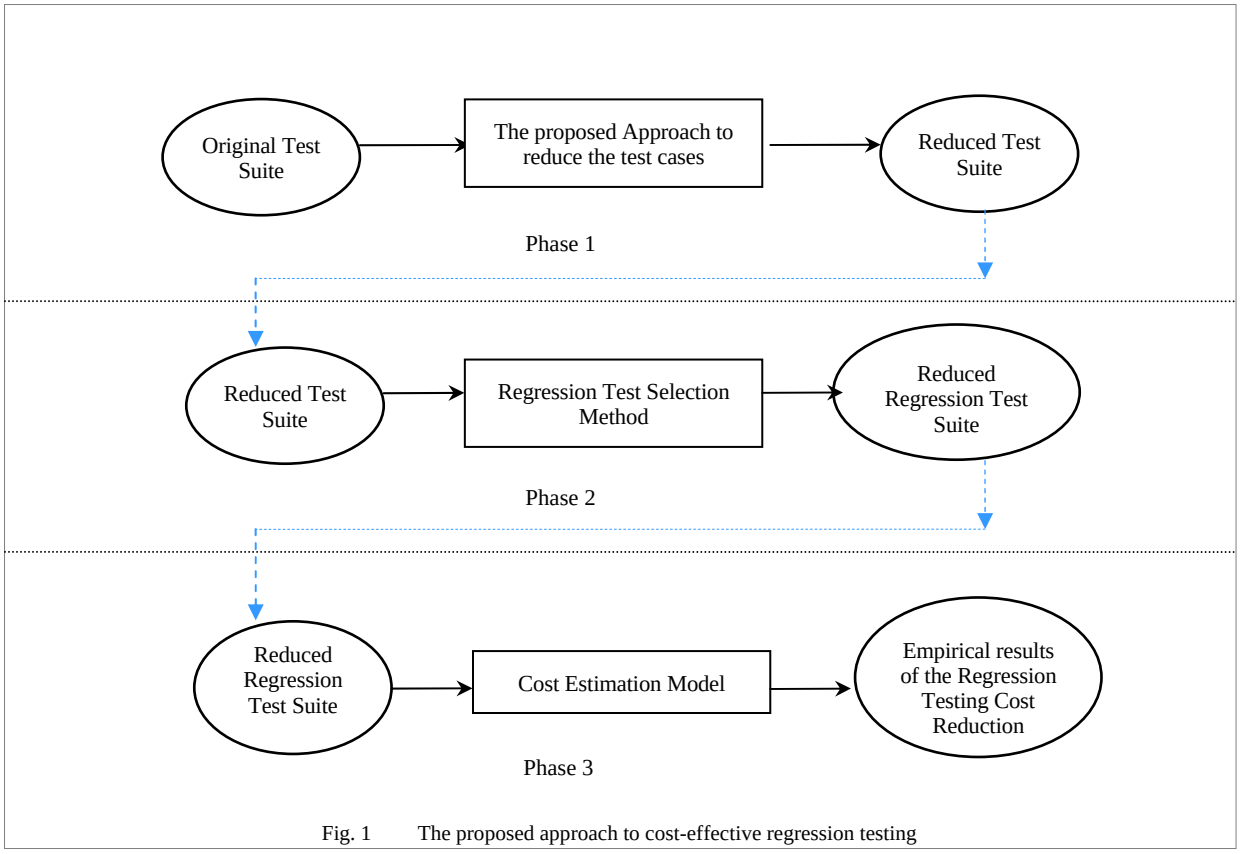

Fig. 1 The proposed approach to cost-effective regression testing

The Phase 1 (Fig.1) of the approach contains the following four steps:

1. View the two aspects that is functionality and boundary value testing together
2. Identify the situation(s) (considering functionality and boundary values) which can be tested in single test case(s) so as to design minimal test cases
3. Proving logically that the single test case(s) in-fact covering both the aspects.
4. Applying above three steps to case studies and validating

By applying the above mentioned approach we get the "Reduced Test Suite" that covers the same functionality of the software as the original test suite. This is validated in the case studies.

Phase 2: Deriving the "Reduced Regression Test Suite"

Regression testing process involves selecting a subset of the test cases from the original test suite, and if necessary creates some new test cases to test the modified software.

Let P is the original software product, P' is the modified software product and T is the set test cases to test P. A typical regression testing on modified software proceeds as follows:

A. Select $T' \subseteq T$, a set of test cases to execute on the modified software product P'.

B. Test P' with T', to verify modified software product's correctness with respect to T'.

C. If necessary, create T'', a set of new test cases to test P'.

D. Test P' with new tests T'', to verify P' correctness with respect to T''.

In Phase 1 (Fig 1), the "Reduced Test Suite" is derived. In Phase 2 (Fig 1), the "Reduced Regression Test Suite" is derived by applying the regression test selection method shown in the Figure 2. This regression test select ion method contains the following 3 steps:

1. Select a subset of test cases from the reduced test suite (derived in Phase1) which covers the major functionality of the product.
2. Select test cases that cover the scenarios to test the bug fixes included in the regression build

3. Create new test cases, to test the (if any) new enhancements included in the regression build.

In step1 of this approach, we are selecting subset of test cases from the reduced test suite. So, this selected subset will also contain the less number of tests as compared to the subset selected from the original test suite. This reduced regression test suite covers the same functionality as the original regression test suite that is derived without applying our approach.

The reduced regression test suite derived using this approach is empirically evaluated in the 'case studies' section of the paper.

Phase 3: Regression Testing Cost Estimation

In Phase 3 of the proposed approach we calculate the estimated reduction in regression testing achieved by using the proposed approach. The authors proposed an approach to cost estimation in black-box testing environment in [19]. Using this approach the regression testing in black-box environment involves the following major activities.

- Environment setup for testing (T_{env})
- Verification of the fixed bugs which were reported in the previous testing cycle (T_{bv})
- Test Suite execution (T_e)
- Test Report Generation (T_{rg})
- Test Report Analysis (T_{ra})
- Reporting the Bugs (T_{br})

As the above mentioned actives are performed on an each and every build, they occupies major portion of the overall regression testing time. The time required to complete regression testing on one intermediate or regression build is calculated using the following equation.

$$E_{ib} = T_{env} + ((N_t \times T_e)/60) + T_{rg} + T_{ra} + T_{bv} + T_{br} \quad (1)$$

where, the 'T_e' indicates the average time required to execute a single test case and the 'N_t' is the total number of the test cases executed for that particular regression testing cycle.

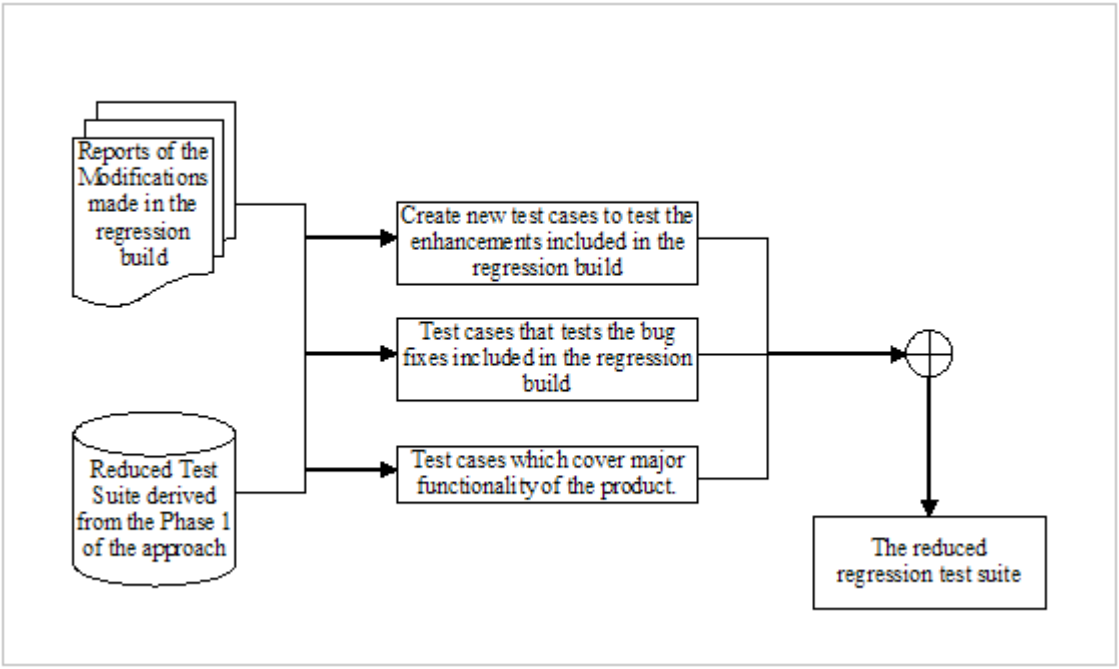

Fig. 2 The regression test case selection

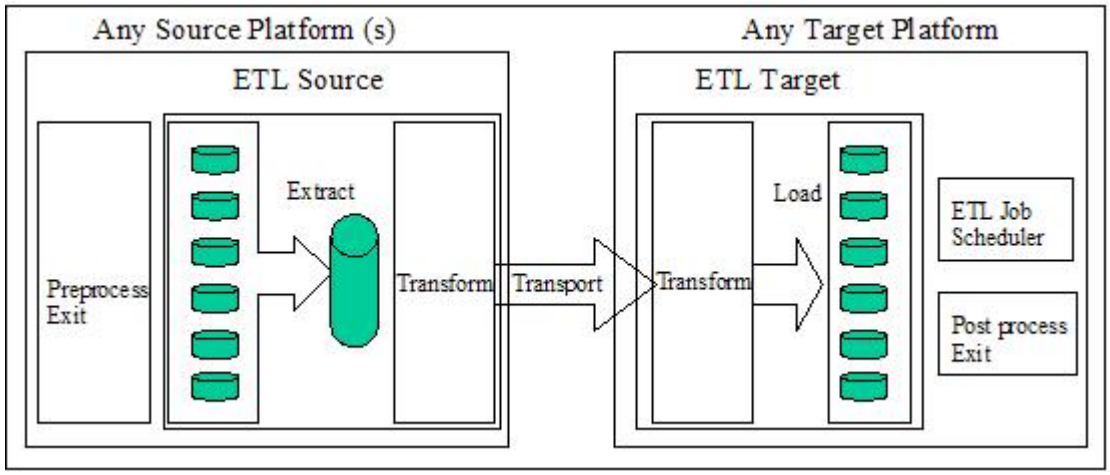

Fig. 3 The ETL process

The equation (1) gives the estimated effort required to test one regression build, in man-hours. The estimated the regression testing cost on a regression build can be calculated using the following equation.

$$Ctotal = Se \times Etotal \quad (2)$$

where, ' *Se* ' is the average salary paid to a testing engineer per man-hour.

The salary paid to the employee per man-hour mainly depends on the organization and geography of the employee. So, the estimated regression testing cost for the product can be calculated based on these factors and using equation (2).

The following section describes the empirical validation of the proposed approach.

4. Empirical Studies and Results

The proposed approach is applied on four real-time ETL tool (Data ware housing tool) components: DB2 ETL DB Component, Sybase ETL DB Component, Teradata ETL DB Component and MySQL ETL DB Component. Concepts explained in Fig. 3 and Fig. 4, are generic and applicable to all the above four case studies. In Fig. 3, ETL, which stands for "extract, transform and load", is the set of functions combined into one tool or solution that enables companies to "extract" data from numerous databases, applications and systems, "transform" it as appropriate, and "load" it into another databases, a data mart or a data warehouse for analysis, or send it along to another operational system to support a business process.

The phase 1 of the approach is applied to the case studies as given below:

Phase 1: Deriving the "Reduced Test Suite"

The test suite that tests the complete functionality of an ETL tool include: Functional test cases (T_f), Boundary Value test cases (T_b), Stress test cases (T_s), Performance test cases (T_p) and other test cases (T_o) like negative test cases. So the Total Number of test cases (T_n) are:

$T_n = T_f + T_b + T_s + T_p + T_o$

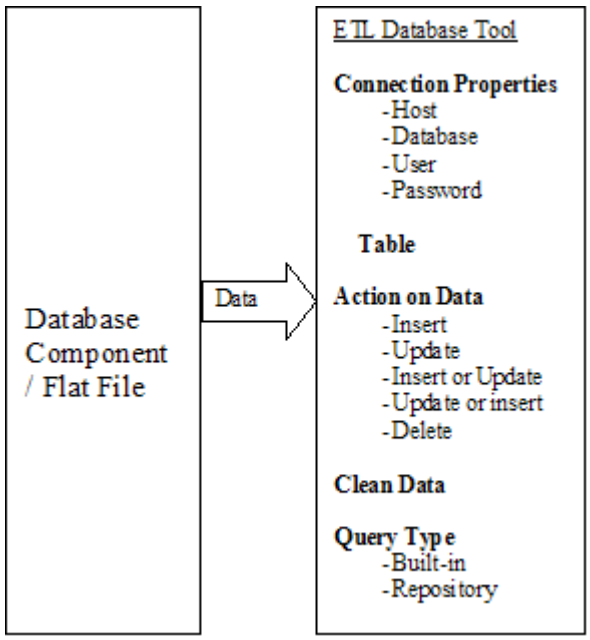

Fig. 4 The ETL Database Component write process

TABLE1. FUNCTIONAL TEST CASES BEFORE APPLYING THE PROPOSED APPROACH OF PHASE 1

Test Case ID	Description	Preconditions	Expected Result	Test Status	Comments
TC*f*1	Test on writing the data to the target table with Action on data = Insert		The job should add new rows to the target table and stop if duplicate rows are found.		
TC*f*2	Test on writing the data to the target table with Action on data = Update		The job should make changes to existing rows in the target table with the input data.		
TC*f*3	Test on writing the data to the target table with Action on data = Insert or Update		The job should add new rows to the target table first and then update existing rows.		
TC*f*4	Test on writing the data to the target table with Action on data =Update or Insert		The job should update existing rows first and then add new rows to the target table.		
TC*f*5	Test on writing the data to the target table with Action on data =Delete		The job should remove rows from the target table corresponding to the input data.		

The Fig. 4 shows some attributes of a generalized ETL Database Component write process. In this write process, the source could be an ETL DB Component or a flat file and the target is a ETL DB Component.

In the write process, the target ETL DB Component reads data from the source component, connects to the respective database using the connection properties specified and writes that data in to the target table.

The test case design using the phase 1 of proposed approach, for DB2 ETL DB Component is described in section A.

A. DB2 ETL DB Component Test Case Design

The Fig. 5 shows the metadata of the table 'sampletable' used in the DB2 ETL DB Component case study. This is a DB2 table that contains 5 columns. The col1 is integer type, col2 is character type, col3 is varchar type, col4 is decimal type and col5 is date type.

The Table 1 shows some sample Functional test cases for the DB2 ETL DB Component write process. Each of these test cases tests a single functionality or scenario of the DB2 ETL DB Component to ensure the particular attribute or function is working properly.

Column name	Datatype schema	Data type name	Column Length	Scale	Nulls
COL1	SYS	INTEGER	4	0	No
COL2	SYS	CHARACTER	9	0	Yes
COL3	SYS	VARCHAR	9	0	Yes
COL4	SYS	DECIMAL	12	3	Yes
COL5	SYS	DATE	4	0	Yes

Fig. 5 Metadata of the sample table

The Table 2 shows some sample Boundary Value test cases for the DB2 ETL DB Component write process. Each of these test cases tests a single column or data type to ensure the boundary values of that data type are written properly to the target table.

The test case design for DB2 ETL DB Component using the proposed approach of phase 1 is described in the following four sub sections (A.1 – A.4).

TABLE 2. BOUNDARY VALUE TEST CASES BEFORE APPLYING THE PROPOSED APPROACH OF PHASE 1

Test Case ID	Description	Preconditions	Expected Result	Test Status	Comments
TC*b*1	Test on writing the data to col1 with INTEGER data type boundary values		The job should read the INTEGER data type boundary values from input data and write to the target table successfully.		
TC*b*2	Test on writing the data to col2 with CHAR data type boundary values		The job should read the CHAR data type boundary values from input data and write to the target table successfully.		
TC*b*3	Test on writing the data to col3 with VARCHAR data type boundary values.		The job should read the VARCHAR data type boundary values from input data and write to the target table successfully.		
TC*b*4	Test on writing the data to col4 with DOUBLE data type boundary values		The job should read the DOUBLE data type boundary values from input data and write to the target table successfully.		

TC*b*5	Test on writing the data to col5 with DATE data type boundary values		The job should read the DATE data type boundary values from input data and write to the target table successfully.		

A.1. View the two aspects together (Step 1)

Many software testing techniques are required to test functionality of a software product completely. A large number of test cases are generated by applying the various testing techniques. These test cases include: functional test cases (Tf), Boundary Value test cases (Tb) , Stress test cases (Ts), Performance test cases (Tp) and other test cases (To) like negative test cases.

Tn = Tf + Tb+ Ts + Tp+ To.

Most of the test cases in this test suite belong to test cases that test the functionality and boundary values of the product. The proposed approach in Phase1 is focused to reduce test cases considering test cases that test functionality and boundary values.

A.2. Identifying the situations that can be tested in a single test case and designing minimized test case set (Step 2)

The test case TCf1 tests the functionality of the DB2 ETL DB Component when the attribute 'Action on Data' is set to 'Insert' and the test case TCb1 tests the INTEGER data type boundary value that is written to the target DB2 table. Both of these test cases TCf1 and TCb1 are testing the two aspects i.e. functionality and boundary values of the DB2 ETL DB Component.

By using the proposed approach in phase1 these two test cases could be viewed together and tested in a single test case. For example, the test cases TCf1 and TCb1 are viewed together and designed a single test case TCm1 (Table 3) that covers the both aspects. The minimized test case set designed using the proposed approach in phase 1 is shown in the Table 3.

A.3. Providing logically that the single test case in fact covers both the aspects (Step 3)

Each test case in the minimized test case set described in Table 3 will test the functionality of the DB2 ETL DB Component to ensure that the particular attribute is working properly and also tests the boundary values for various columns in the target table to ensure that the boundary values of that column data type are written properly. For example, the TCm1 in the minimized test case set tests whether the DB2 ETL DB Component is working properly when the attribute 'Action on Data' is set to 'Insert' and also tests whether the INTEGER data type boundary value is written to the target table properly which were tested by the test cases TCf1 and TCb1.

In similar way, the remaining test cases in the minimized test case set {TCm1 – TCm5} described in Table 3 will test the both aspects, functionality and the boundary values of DB2 ETL DB Component which have been tested by the test cases {TCf1-TCf5 and TCb1-TCb5}.

A.4. Applying the above three steps to case studies and validating (step 4)

If the number of boundary value test cases that are viewed together with functional test cases, the number of test cases test cases reduced is T_{br}. Then, after applying the phase 1 of the proposed approach, the total number of test cases is minimized to:

$T_{\mathbf{min}} = T_{\mathbf{n}} - T_{br}$

And, the percentage of test case reduction (T_{red} %) is:

$T_{red}\ \% = ((T_n - T_{min}) / T_{\mathbf{n}}) * 100$

TABLE 3. THE MINIMIZED TEST CASE SET DESIGNED USING THE PROPOSED APPROACH IN PHASE 1

Test Case ID	Description	Preconditions	Expected Result	Test Status	Comments
TC*m*1	Test on writing the data to the target table with Action on data = Insert and col1 contains INTEGER data type boundary values		The job should read the input data, add new rows to the target table successfully and stop if duplicate rows are found.		
TC*m*2	Test on writing the data to the target table with Action on data = Update and col2 contains CHAR data type boundary values		The job should read the input data and make changes to existing rows in the target table with the input data		
TC*m*3	Test on writing the data to the target table with Action on data = Insert or Update and col3 contains VARCHAR data type boundary values		The job should read the input data, add new rows to the target table first and then update existing rows		
TC*m*4	Test on writing the data to the target table with Action on data = Update or Insert and col4 contains DOUBLE data type boundary values		The job should read the input data, update existing rows first and then add new rows to the target table		

TC*m*5	Test on writing the data to the target table with Action on data = Delete and col5 contains DATE data type boundary values		The job should read the input data and remove rows from the target table corresponding to the input data		

TABLE 4. REDUCED REGRESSION SUITE

ETL DB Component	Original Test Suite (Tn)	Reduced Test Suite – Phase 1 (Tmin)	Original Regression Suite (TR)	Reduced Regression Suite- Phase 2 (TRmin)
DB2 ETL DB Component	3563	2609 (26.7 %)	1846	1304
Sybase ETL DB Component	2968	2079 (29.98 %)	1497	1034
Teradata ETL DB Component	4234	2798 (33.91 %)	2534	1624
MySQL ETL DB Component	3657	2484 (32.07 %)	1668	1166

In similar way, the proposed approach is also applied on Sybase ETL DB Component, Teradata ETL DB Component and MySQL ETL DB Component. The second column of Table 4 describes the total number of test cases (T_n) before applying phase 1 of the proposed approach, the third column describes the total number of test cases in the minimized test case suite (T_{min}) after applying the phase 1 of the proposed and the percentage of test case reduction (T_{red} **%**), given in parenthesis.

After applying the proposed approach in phase 1, the total number of test cases for DB2 ETL DB Component, Sybase ETL DB Component, Teradata ETL DB Component and MySQL ETL DB Component test cases are reduced by 34 %,27 %,30 % and 32 % respectively. The results indicate that the number of test case reduction is ranging between 27 to 34 percent (Table 4, 3rd column). Hence the Phase 1 of the proposed approach is validated through case studies.

Phase 2: Deriving the "Reduced Regression Test Suite"

Regression testing is a critical part of the software maintenance that is performed on the modified software to ensure that the modifications do not adversely affect the unchanged portion of the software.

Using the proposed approach for regression test selection, we have selected a subset of test cases from the reduced test suite (derived in Phase1) which covers the major functionality of the product, selected test cases that cover the scenarios to test the bug fixes included in the regression build, and created new test cases, to test the (if any) new enhancements included in the regression build. This derived "Reduced Regression Test Suite" covers the same functionality of the software product as the regression suite that is derived from the original test suite (without reduction).

The phase 2 of the approach is applied on four case studies and the results are recorded in Table 4. The fourth column in table 4 describes the number of regression test cases (T_R) that are derived by applying the proposed regression test selection method on the original test suite (i.e before applying the Phase1 of the proposed approach). The fifth column in Table 4 describes the "Reduced Regression Test Suite" (T_{Rmin}) which is derived by applying the proposed regression test selection method on the "Reduced Test Suite" derived in Phase1.

This reduction is independent of the regression test selection method that is used to select the regression test cases. If the number of test cases in the original test suite is reduced, then subsequently the number of regression test cases also reduced.

Phase 3: Regression Testing Cost Estimation

The table 5 presents the required average effort for each of the testing activities in black-box testing, based the historical data derived from analyzing 40 completed software projects [19].

TABLE 5. AVERAGE TIME REQUIRED FOR TESTING ACTIVIIES

Testing activity	Avg. Estimated effort
Environment setup for testing	3 Hrs
Verification of the fixed bugs	20 min / bug
Test Suite execution	1.2 min / test case
Test Report Generation	9 min
Test Report Analysis	20 min

Reporting the Bugs	18 min / bug

The estimated effort required to complete the testing on one regression build calculated using the equation (1) is:

For original regression test suite:

E_{ib} = 3 + ((1864 x 1.2) +9+20+4x20+4x18) / 60 = 43.29 Hrs

For reduced regression test suite:

E_{ib} = 3 + ((1304 x 1.2) +9+20+4x20+4x18) / 60 = 32.09 Hrs

According to C. Jones [18] the average salary paid to a software engineer is $100 per hour. The total estimated cost for testing the complete product before it gets released to the customer is calculated using the equation (2):

For original regression rest Suite:

C_{total} = 100 x 43.29 = 4329 $

For reduced regression test Suite:

C_{total} = 100 x 32.09 = 3209 $

So, the estimated regression testing cost of the 'DB2 ETL DB Component' using the original regression suite is 4329 $, and the estimated regression testing cost of the 'DB2 ETL DB Component' using the reduced regression suite is 3209 $. In Table 6, the 4th column indicates the estimated regression testing cost using the original regression test suite, and the 5th columns indicates the estimated regression testing cost using the reduced regression test suite. For the remaining three projects the regression testing costs are estimated using the proposed approach and the final results are given in the table 6.

The average salary paid to a software engineer varies based on the organization and the geography location. As we have estimated the exact amount of effort required, the project manager could easily estimate the exact testing cost using equation (2), by substituting average salary paid to the employee in their organization.

The regression testing cost reduced by applying the proposed approach is:

$C_{Rred} = C_R - C_{Rmin}$

The percentage of reduction in regression testing cost is: $C_{Rred}\ \% = ((C_R - C_{Rmin}) / C_R) * 100$

The regression testing cost reduced for 'DB2 ETL DB Component' calculated using the above equation is:

$C_{Rred}\ \% = ((4329-3209)/ 4329)*100 = 25.87\ \%$

The percentage of reduction in regression testing cost (C_{Rred} %) by using the proposed approach, on one regression testing cycle, for various projects calculate using the above equations are shown in the 6th column of the Table 6.

The regression testing needs to be performed on many intermediate software builds of the product during the software maintenance phase.

Let B_n {n=1,2,3,…,12} is the number of builds for a particular month on which the regression testing needs to done.

Then the total number of builds per year is $\sum_{n=1}^{12} B_n$, and the average number of builds per month is $\frac{1}{12} \times \left(\sum_{n=1}^{12} B_n \right)$.

So, the regression testing cost reduced per month is $(C_{Rred}\%) \times \frac{1}{12} \times \left(\sum_{n=1}^{12} B_n \right)$, and

per year is $(C_{Rred}\%) \left(\sum_{n=1}^{12} B_n \right)$.

TABLE 6. ESTIMATED REGRESSION TESTING COST REDUCTION

ETL DB Component	Original Regression Suite (T_R)	Reduced Regression Suite- Phase 2 (T_{Rmin})	Estimated Cost to test the original Regression suite (T_R)	Estimated Cost to test the Reduced Regression Suite (T_{Rmin})	Percentage of reduced Regression testing cost (T_{Rmin})
DB2 ETL DB Component	1846	1304	4329	3209	25.87 %
Sybase ETL DB Component	1497	1034	3595	2669	25.75 %
Teradata ETL DB	2534	1624	5669	3849	32.10 %

Component					
MySQL ETL DB Component	1668	1166	3637	2933	19.35 %

By applying the proposed approach, C_{Rred}% percent regression testing cost is reduced for a ETL DB Component. These case studies show that, the proposed approach saves a substantial amount of regression testing time and effort. The cost of the regression testing for DB2 ETL DB Component, Sybase ETL DB Component, Teradata ETL DB Component and MySQL ETL DB Component is reduced by 25.87 %, 25.75 %, 32.10 % and 19.35 % respectively. The results indicate that by applying the proposed approach, the reduction in cost of regression testing is ranging between 19.35 to 32.10 percent (Table 6, 6th column).

5. Conclusions and Future work

The proposed approach reduces the number of regression test cases in black box environment, independent of the regression test selection methods that are available. The effort required to apply this approach is a one-time effort, but it reduces the effort and time required for all the remaining regression testing cycles of the software.

The proposed approach is applied on four real-time ETL Tools (Data ware housing tools) that are used by many customers all over the world. The tested ETL tool components are DB2 ETL DB Component, Sybase ETL DB Component, Teradata ETL DB Component and MySQL ETL DB Component. It is found from the case studies that the cost of regression testing can be reduced by applying the proposed method and the reduction in regression testing cost is ranging between 19.35 and 32.10 percent. Hence, by using the proposed approach the regression testing cost can be reduced considerably.

As part of the future work, we are planning to propose an enhanced regression test selection method in black-box environment which further reduces the regression testing cost.

References

[1] IEEE Std 610.12-1990, IEEE Standard Glossary of Software Engineering Terminology.

[2] James A. Jones and Mary Jean Harrold, “Test-Suite Reduction and Prioritization for Modified Condition/Decision Coverage”, IEEE Transactions on Software Engineering, Vol. 29, Issue. 3, March 2003.

[3] Saif-ur-Rehman Khan Nadeem, A.Awais, “TestFilter: A Statement-Coverage Based Test Case Reduction Technique”, IEEE Multitopic Conference, page(s): 275 - 280, 23-24 Dec. 2006.

[4] T. Y. Chen and M. F. Lau, “Dividing strategies for the optimization of a test suite”, Information Processing Letters, 60(3):135–141, Mar. 1996.

[5] M. J. Harrold, R. Gupta, and M. L. Soffa, “A methodology for controlling the size of a test suite”, ACM Transactions on Softw.Eng. and Meth., 2(3):270–285, July 1993.

[6] W. E.Wong, J. R. Horgan, S. London, and A. P.Mathur, “Effect of test set minimization on fault detection effectiveness”, 17th international conference on Software engineering, pages 41 – 50, 1995.

[7] G. Rothermel, R.H. Untch, C. Chu, and M.J. Harrold, “Prioritizing Test Cases for Regression Testing,” IEEE Trans. Software Eng., vol.27, no. 10, pp. 929-948, Oct. 2001.

[8] H. K. N. Leung and L. White, “A cost model to compare regression test strategies”, In Proc. Conf. Softw. Maint., pages 201–208, Oct. 1991.

[9] G. Rothermel and M. J. Harrold, “A safe, efficient regression test selection technique”, ACM Transactions on Software Engineering Meth.,6(2):173–210, April 1997.

[10] B. Beizer. Software Testing Techniques. VanNostrand Reinhold, New York, NY, 1990.

[11] H. K. N. Leung and L. White. “Insights into regression testing”, In Conf. Softw. Maint., pages 60–69, October 1989.

[12] M. Jean Harrold, Rajiv Gupta,Mary Lou Soffa, "A methodology for controlling the size of a test suite, ACM Transactions on Software Engineering and Methodology, Volume 2, Issue 3, 1993.

[13] Zhenyu Chen, Baowen Xu, Xiaofang Zhang, Changhai Nie, "A novel approach for test suite reduction based on requirement relation contraction", Proceedings of the 2008 ACM symposium on Applied computing,Pages 390-394,2008

[14] S. Parsa, A. Khalilian and Y. Fazlalizadeh, "A New Algorithm to Test Suite Reduction Based on Cluster Analysis",iccsit, pp.189-193, 2009 2nd IEEE International Conference on Computer Science and Information Technology, 2009..

[15] Pravin M. Kamde, V. D. Nandavadekar, R. G. Pawar, “Value of Test Cases in Software Testing”, International Conference on Management of Innovation and Technology, IEEE, 2006.

[16] G. Rothermel, M.J. Harrold, J. Ostria, and C. Hong, “An Empirical Study of the Effects of Minimization on the Fault Detection Capabilities of Test Suites”, Proc. Int’l Conf. Software Maintenance, PP. 34-43, Nov. 1998.

[17] Kiran Kumar J, A. Anada Rao, M. Gopi Chand, K. Narender Reddy, “An Approach to test case Design for cost effective Software Testing”, IMECS-IAENG-2009.

[18] S. Schach, Software Engineering. Boston: Aksen Assoc., 1992.

[19] Kiran Kumar J and Prof. A. Ananda Rao, "An Approach to Software Testing Cost Estimation in Black-Box

Environment", International Journal of Electrical, Electronics and Computer Systems, April 2011.
[20] H. Agrawal, J. Horgan, E. Krauser, and S. London, “Incremental Regression Testing,” Proc. Conf. Software Maintenance, pp. 348–357, Sept. 1993.
[21] T. Ball, “On the Limit of Control Flow Analysis for Regression Test Selection,” Proc. Int’l Symp. Software Testing and Analysis, ISSTA, Mar. 1998.
[22] S. Bates and S. Horwitz, “Incremental Program Testing Using Program Dependence Graphs,” Proc. 20th ACM Symp. Principles of Programming Languages, Jan. 1993.
[23] P. Benedusi, A. Cimitile, and U. De Carlini, “Post-Maintenance Testing Based on Path Change Analysis,” Proc. Conf. Software Maintenance, pp. 352–361, Oct. 1988.
[24] D. Binkely, “Semantics Guided Regression Test Cost Reduction,” IEEE Trans. Software Eng., vol. 23, no. 8, Aug. 1997.
[25] Y.F. Chen, D.S. Rosenblum, and K.P. Vo, “TestTube: A System for Selective Regression Testing,” Proc. 16th Int’l Conf. Software Eng., pp. 211–222, May 1994.
[26] K.F. Fischer, “A Test Case Selection Method for the Validation of Software Maintenance Modification,” Proc. COMPSAC’77, pp.421–426, Nov. 1977.
[27] K.F. Fischer, F. Raji, and A. Chruscicki, “A Methodology for Retesting Modified Software,” Proc. Nat’l Telecommunications Conf., pp. 1–6, Nov. 1981.
[28] R. Gupta, M.J. Harrold, and M.L. Soffa, “An Approach to Regression Testing Using Slicing,” Proc. Conf. Software Maintenance, pp.299–308, Nov. 1992.
[29] M.J. Harrold and M.L. Soffa, “An Incremental Approach to Unit Testing During Maintenance,” Proc. Conf. Software Maintenance,pp. 362–367, Oct. 1988.
[30] M.J. Harrold and M.L. Soffa, “An Incremental Data Flow Testing Tool,” Proc. Sixth Int’l Conf. Testing Computer Software, May 1989.
[31] J. Hartmann and D.J. Robson, “RETEXT—Development of a Selective Revalidation Prototype Environment for Use in Software Maintenance,” Proc. 23rd Hawaii Int’l Conf. System Sciences, pp. 92–101, Jan. 1990.
[32] J. Hartmann and D.J. Robson, Techniques for Selective Revalidationk” IEEE Software, vol. 16, no. 1, pp. 31–38, Jan. 1990.
[33] J. Laski and W. Szermer, “Identification of Program Modifications and Its Applications in Software Maintenance,” Proc. Conf. Software Maintenance, pp. 282–290, Nov. 1992.
[34] J.A.N. Lee and X. He, “A Methodology for Test Selection,” J. Systems and Software, vol. 13, no. 1, pp. 177–185, Sept. 1990.
[35] H.K.N. Leung and L. White, “Insights into Regression Testing,” Proc. Conf. Software Maintenance, pp. 60–69, Oct. 1989.
[36] H.K.N. Leung and L. White, “Insights into Testing and Regression Testing Global Variables,” J. Software Maintenance, vol. 2, pp. 209–222, Dec. 1990.
[37] H.K.N. Leung and L.J. White, “A Study of Integration Testing and Software Regression at the Integration Level,” Proc. Conf. Software Maintenance, pp. 290–300, Nov. 1990.
[38] T.J. Ostrand and E.J. Weyuker, “Using Dataflow Analysis for Regression Testing,” Proc. Sixth Ann. Pacific Northwest Software Quality Conf., pp. 233–247, Sept. 1988.
[39] B. Eherlund and B. Korel, “Modification Oriented Software Testing,” Conf. Proc.: Quality Week, pp. 1–17, 1991.
[40] B. Sherlund and B. Korel, “Logical Modification Oriented Software Testing,” Proc. 12th Int’l Conf. Testing Computer Software, June 1995.
[41] A.B. Taha, S.M. Thebaut, and S.S. Liu, An Approach to Software Fault Localization and Revalidation Based on Incremental Data Flow Analysis,” Proc. 13th Ann. Int’l Computer Software and Applications Conf., pp. 527–534, Sept. 1989.
[42] F. Vokolos and P. Frankl, “Pythia: A Regression Test Selection Tool Based on Textual Differencing,” Proc. Third Int’l Conf Reliability, Quality, and Safety of Software Intensive Systems, ENCRESS’97, May 1997.
[43] L.J. Wshite and H.K.N. Leung, “A Firewall Concept for Both Control- Flow and Data-Flow in Regression Integration Testing,” Proc. Conf. Software Maintenance, pp. 262–270, Nov. 1992.
[44] L.J. White, V. Narayanswamy, T. Friedman, M. Kirschenbaum, P.Piwowarski, and M. Oha, “Test Manager, A Regression Testing Tool,” Proc. Conf. Software Maintenance, pp. 338–347, Sept. 1993.
[45] S.S. Yau and Z. Kishimoto, “A Method for Revalidating Modified Programs in the Maintenance Phase,” COMPSAC’87: Proc. 11th Ann. Int’l Computer Software and Applications Conf., pp. 272–277, Oct. 1987.

Prof. Ananda Rao Akepogu received B.Sc. (M.P.C) degree from Silver Jubilee Govt. College, SV Univer-sity, Andhra Pradesh, India. He received B.Tech. degree in Computer Science & Engineering and M.Tech. degree in A.I & Robotics from University of Hyderabad, Andhra Pradesh, India. He received Ph.D. from Indian Institute of Technology, Madras, India. He is Professor of Computer Science & Engineering and Principal of JNTU College of Engineering, Anantapur, India. Prof. Ananda Rao published more than fifty research papers in international journals, conferences and authored three books. His main research interest includes software engineering and data mining.

Kiran Kumar J is pursuing Ph.D. in Computer Science & Engineering from JNTUA, Anantapur, India and he received his M.Tech. in Computer Science & Engineering from the same university. He received B.E. degree in Computer Science & Engineering from Amaravati University, India. He has received the “Teradata Certified Master” certification form the Teradata. Currently he is working for IBM India Software Labs in the area of Software Testing since 2005. His main research interests include software engineering and Software Testing. He is a member of IEEE, ACM and IAENG.