2011

# Change Impact Analysis: A Tool for Effective Regression Testing

Prem Parashar
*Sheridan College*, prem.parashar@sheridancollege.ca

Rajesh Bhatia
*Thapar University*

Arvind Kalia
*Himachal Pradesh University*

# Change Impact Analysis: A Tool for Effective Regression Testing

Prem Parashar[1], Rajesh Bhatia[2], and Arvind Kalia[3]

[1] Computer Science Department, Chitkara University, Barotiwala, Solan, India
[2] Computer Science Department, Thapar University, Patiala, India
[3] Computer Science Department, Himachal Pradesh University, Shimla, India
{prem.parashar,rbhatiapatiala,arvkalia}@gmail.com

**Abstract.** Change impact analysis is an imperative activity for the maintenance of software. It determines the set of modules that are changed and modules that are affected by the change(s). It helps in regression testing because only those modules that are either changed or affected by the suggested change(s) are re-tested. Change impact analysis is a complex activity as it is very difficult to predict the impact of a change in software. Different researchers have proposed different change impact analysis approaches that help in prioritization of test cases for regression testing. In this paper, an approach based on Total Importance of Module (TIM) has been proposed that determines the importance of a module on the basis of (i) user requirements, and (ii) system requirements. The results of the proposed algorithm showed that the importance of a module is an essential attribute in forming a prioritized test suite for regression testing.

**Keywords:** granularity, prioritization, maintenance, change impact, impact set.

## 1 Introduction

Software maintenance is a regular activity as the requirements of user changes frequently. The maintenance of software may be preventive maintenance, adaptive maintenance or extensive maintenance. Regression testing is one of the most important and complex software testing methods [2] because even a subtle change in software may require rerun of all its old test cases and test cases designed for the change [16]. The major challenge for regression testing is the permissible time budget [8, 10]. After maintenance, modified software is required to be re-installed at the earliest. Due to a limited time budget allowed for regression testing, it may not be possible to rerun the entire test suite designed for the original software and test suite designed for its modified components. For gaining confidence in maintenance, all those test cases that cover the changed components and the components that get affected by the changes should be rerun[21]. The impact set of modules generated after the suggested change(s) may be formulated with the help of change impact analysis. Change impact analysis helps in finding the changed modules and modules affected by the change. A fine grained change impact analysis technique is always preferred to generate the precise impact set [13]. The impact set generated thus facilitates in effective regression testing because only the modules that constitute the impact set are tested.

S. Dua, S. Sahni, and D.P. Goyal (Eds.): ICISTM 2011, CCIS 141, pp. 160–169, 2011.

Several change impact analysis approaches are reported in the review of literature [20, 21] . Most of the approaches are based on two assumptions, i.e. (i) all modules have equal importance in software [12], and (ii) each fault has equal severity [3]. In real practice, the importance of a module may play a vital role in regression testing i.e. module with the highest priority should be tested first. This is the main motivation factor behind performing this study.

In this study, it has been assumed that (i) user requirements, (ii) the call graph of system, and (iii) the historical records about the execution of the module are three main factors that define the overall importance of module. To determine the importance of a module, three metrics i.e. User-defined Importance of Module (UIM), Dynamic Importance of Module (DIM), and History Value of Module (HVM) have been proposed. UIM is supposed to be defined by user in requirements, DIM of a module is supposed to be dependent on the structure of the call graph of system, and HVM is defined by the historical records of the execution of the module with respect to the execution of the system.

## 2 Review of Literature

Rajlich [13] et al. proposed an iterative impact analysis method that uses the different level of granularity in order to achieve high level of precision. The granularity of components was performed at three different levels i.e. class, class member and code fragment level. From the empirical study conducted, it was evident that proposed algorithm is effective when the level of granularity is grained to code fragment level. For dynamic change impact analysis of software, atomic changes in source code responsible for the behavioral changes of test cases were considered .The main concern of this technique is to identify a subset of the changes that actually affects the behavior of a test case [15]. The results of the case studies based upon above technique evident that most of changes were captured by the execution of half of test cases.

Tie [19] et al. proposed a component interaction trace based approach to trace out the dynamic change impact analysis. The static structure of the components and their interfaces through which they are integrated is represented with the help of collaboration diagram and the dynamic model is represented with the help of the sequence diagram describing the objects interactions. If the two interfaces are serving approximately same purposes then it was proposed to replace one interface by the other and the unique services provided by the former were introduced in the later.

Sherriff [17,18] et al. proposed a change impact analysis approach based upon singular value decomposition (SVD) to find the structure of the file association clusters and the amount of variation done by this cluster in the original system after a change. They explored the concept of SVD in finding the impact of change in a system that contains a number of executable and non-executable files. This technique has been found quite satisfactory if the level of granularity is file and provides encouraging results. This technique is suitable for finding file association clusters of source files as well as non-source files. For object-oriented software, a model-based approach [10] that takes two different models i.e. activity diagram and sequence

diagram to establish mapping is used to find the impact of change. These models of object-oriented software are mapped on the basis of some well-founded rules of relationship. The main advantage of this approach is that it does not require source code of software.

Rao [14] et al. proposed a quantitative method to detect the impact set if a change is made in an artifact and also to detect an artifact that is affected by the change. The method uses design change propagation probability (DCPP) matrix. DCPP is based upon the degree of coupling between two artifacts. The DCPP method is a quantitative method. It helps in finding the propagation of change. URA used in this case along with the corresponding DCPP clearly depicts the static behavior of the system. The method can also be used to automate the change impact propagation.

Park [12] et al. proposed a method to identify multi dimensional dependencies on the basis of process slicing. Process slicing can handle multi-dimensional dependencies effectively. The proposed algorithm handles the cases where any change in one activity produces change in two or more artifacts. The main objective of the proposed algorithm is to find out and to test that part of the software which is affect by the specified change. The time budget for regression testing is usually less therefore it is not practical to rerun all the tests of the old software along with the additional tests for the changed part. German [5] et al. proposed a method of determining the impact of historical changes on a particular code segment that in terms makes some failures. They considered C language program as a code segment for their study and took all the functions into consideration that affect the performance of the function. They prepare a change impact graph (CIG) that clearly determines all the functions that are falling into the way to make this function affected.

Orso [10] et al. proposed an approach that uses the field data instead of synthesized data to find the change impact analysis and regression testing. The software is tested against the field data collected. In this study, different ways of measuring the impact of change on the user population has been studied. Since live data from the actual users are used to analyze the impact of change, therefore the approach immediately reflects the results whether the changed software is useful to specified class of users or not. The approach is adequate if the user population is limited. Orso [11] et al. performed some empirical studies to make the comparisons of dynamic impact analysis algorithms. In their studies two dependency based algorithms i.e. coverage dynamic impact algorithms and path dynamic impact algorithms are explored to make a comparison about the precision of these algorithms. The empirical studies are performed on small java programs. The level of granularity considered in the studies is of method level. This is one of the empirical studies that deals directly with the cost –benefit analysis. Nadi [9] et al. developed a framework that helps in determining the root cause analysis and change impact analysis for configuration management data bases (CMDB). CMDB is used to store the different types of information like hardware, software and the services provided by an organization. For each component, it records all relevant information. The framework proposed serves two main purposes, (i) it helps to find the root cause of a failure of a component, and (ii) it determines the set of components that are affected by a change in a particular component.

Breech [1] et al. proposed an approach to whole program path-based dynamic change impact analysis. This approach is completely online. The method used in this case assume that any function that is either called directly or indirectly by the changed function or executed after a call to the changed function will constitute impact set. The approach used in this case is safe as it deals with the set of those functions which are potentially affected by a changed function but sometimes it generates imprecise impact set.

Yu and Rajlich [22] studied hidden dependencies among various software components that may cause malfunctioning of the software. They proposed an algorithm that warns about the hidden dependencies in the components that may be caused due to the improper structure of the components. Ma [7] et al. developed an algorithm that helps to find the relative importance of modules present in the system. One of the main objectives of change impact analysis is to trace out those modules that are observing the change or are affected by the proposed change. In real practice, when subtle change is made in one module that affect different parts of the software. The parts may be technical or non technical. After change impact analysis the subset of the modules that have been surfaced out as impact set need to be tested. Due to limited time and other resource constraints it is not easy to test all the modules of the impact set. There is always a tendency from the developer side to test those modules first which detect maximum faults. Ma [7] et al. proposed an effective algorithm that determines the importance of a module to the system. The algorithm successively deals with the hierarchical structure of the module. It deals with the acyclic graph of module effectively.

English [3] et al. studied the fault detection and prediction in open-source software. Their study is mainly content based. They leveraged some popular metrics like Number of methods in a class (NMC), depth of inheritance tree (DIT) number of children (NOC), coupling between object classes (CBO) and response for a class (RFC). The results of the study are encouraging as they strengthen Pareto's law i.e. about 80%of the faults/issues are due to 20% of the code. They have tried to trace out dependent and non-dependent variable to some extent.

Engstrom [4] et al. conducted a longitudinal study on the various techniques used for regression test selection. Since this review contains research papers from the start of the era of software testing, it gives the future researchers as well as developers a systematic review of how the regression testing test selection methods varied from software to software and from time to time. The techniques summarized certainly helps the industry and academia to pursue further research in the area.

## 3 Objectives

The broad objective of this study is to prioritize a test suite on the basis of the importance of changed /affected modules. The specific objectives of the study are:

1. To analyze the significance of the importance of a module when we prioritize a test suite for regression testing.
2. To analyze whether proposed algorithm is effective for prioritization of test cases for regression testing.

## 4 Research Methodology

Change impact analysis is before maintenance activity of software. It gives a clear clue to maintenance manager about (i) whether the required changes should be incorporated in the existing software or (ii) should it be re-engineered. Each module that constitutes software has its own importance. The importance of a module is defined in many ways. In the proposed approach, the importance of a module has been assumed to be defined by the user, by the developer or by the history of its use. For example, ATM system of a bank uses cash withdrawal module most of the time as compared to mini statement module. Therefore according to proposed approach cash withdrawal module is more important as compared to mini statement module. Importance of a module plays a significant role while it is maintained. The most important modules of system should be maintained with utmost care so that at least they should not malfunction at any point of time during execution. At the time of regression testing, those modules that have high priority values to the software are tested first. Total Importance of Module $m_i$ (TIM $(m_i)$), is given by:

$$TIM\ (m_i) = UIM\ (m_i)*DIM\ (m_i)*HVM\ (m_i) \tag{1}$$

In Equation (1), $UIM\ (m_i)$, $DIM\ (m_i)$, and $HVM\ (m_i)$ represent the static importance, dynamic importance, and historical value of the module $m_i$ respectively. *UIM* of a module is a numeric value assigned to it by user. The value ranges from 1 to 5. If a module has a *UIM* value of 5, it means it is the most important, and one means it is the least important. *DIM* of a module is calculated from the Call Graph (CG) of software. It is calculated as:

$$DIM\ (m_i) = TD\ (m_i)/TD\ (CG) \tag{2}$$

In Equation (2), $TD\ (m_i)$ and $TD\ (CG)$ represent total degree of module $m_i$ and total degree of call graph respectively. $HVM\ (m_i)$ is calculated from the historical records of execution of software. It is calculated as:

$$HVM\ (m_i) = THC\ (m_i)/THC\ (CG)*HTI \tag{3}$$

In Equation (3), $THC\ (m_i)$, *THC (CG), and HTI* represent the total number of calls made to module $m_i$ in a history time interval, total calls made to all modules in a history time interval, and a history time interval respectively. For example, if a module is called 10 times in previous five executions and the total modules called during these five executions are 50 then HVM ($m_i$) will be 1.0. *HTI* represents the total number of previous executions taken into account while finding $HVM\ (m_i)$. It is practically difficult to maintain the records of all the previous executions of large software which is running from years. *HTI* limits time interval to some well defined number of executions, so that the better idea of its recent importance could be drawn.

After modification of software, two module sets are generated by change impact analysis i.e. a changed set and an affected set. If a module is changed then its entire neighboring module in the call graph are affected by the change. The neighboring

modules constitute the estimated impact set. Depending upon the importance, changed modules and impacted modules are tested. A simple algorithm for this has been described in Fig. 1. In this algorithm, a term 'cover a module' has been used which signifies that the module has been tested with all test cases required for it, and the bugs have been fixed. Simple mathematical set operations like A U B, A-B have been used to make the proposed algorithm understandable.

```
1.  Let TCM, is impact set and TS, is a test set array where TS[i] contains the number
    of test cases required to cover a module M[i].
2.  Time, is an array where Time[i] stores maximum time required to cover a module
    M[i].
3.  Calculate TIM value for each module of TCM with the help of Equations (1), (2),
    and (3).
4.  Sort the modules of TCM according to TIM values (in descending order)
5.  Rearrange TS, and Time elements according to TCM. e.g. if nth. element of
    original TCM has become first element in the sorted TCM, them Time [1]
    =Time[n] and      TS [1] =TS[n].
6.  Take t=0, i=0      //t represents the time required to execute the selected module(s).
7.  While ((TCM ≠ Φ)and(t ≤ TB)) //TB permissible time budget.
8.   {
9.    if( t+ Time[i]≤ TB)
10.     {
11.       (i)    P  =  P U TS[i]        // P: prioritized test suite
12.       (ii)   t  =  t+ Time[i]
13.       (iii) TCM=TCM-M[i] // module present at i^th location of TCM will be
      removed.
14.     }
15.  Else
16.    TCM=TCM-M[i]
17.          }
```

**Fig. 1.** TIM Test Case Prioritization Algorithm

## 5 Analysis

For the experimental setup, a small menu driven C language program, for basic mathematical operations, has been considered. The program is presented in Fig. 2. Suppose that the line no. 29 (highlighted) of program is modified (modified statement is written in bold and the old statement is written in comments). This change in the program affects the modules main(), sub() and div(). Thus, TCM consists of {main (), sub (), ()}. The CG of this program is represented in Fig. 3.

Module div() (represented with '*') is the changed module and the modules sub(), and main() (represented with '**') are affected modules. Modules add (), and mul() are unaffected by the change. Let us assume that there are six test cases in the test suite: T1, $T_2$, $T_3$, T4, $T_5$, and $T_6$. Let time taken(in seconds) to execute test cases $T_1$, $T_2$, $T_3$,$T_4$, T5, and $T_6$ be 3, 2, 4,1, 5,and 2 seconds, respectively.

```
1.  #include <stdio.h>
2.  void main(void)
3.  {
4.    int a, b, sum,subt , mult,divd ;
5.    char choice;
6.    int add(int,int);    int sub(int,int); int mul(int, int);  int div(int,int);
7.    scanf("%d%d%c", &a,&b, &choice);
8.    switch(choice)
9.   {
10.  case 'a':          printf("Addition of a and b=%d\n", add(a,b)); break;
11.  case 's' :         printf("subtraction of a and b=%d\n",sub(a,b)); break;
12.  case 'm':         printf("Multiplication of a and b=%d\n",mul(a,b)); break;
13.  case 'd' :         printf("division of a by b=%d\n", div(a,b)); break;
14.  default:          printf("Invalid Choice\n");
15.  }
16. }   /* end main()  */
17. int add(int x,int y) {return(x+y);}
18. int sub(int x,int y) { return (x+y);}
19. int mul ( int x, int y)
20. {
21.    int i=1,p=0;
22.    for(i=1;i<=y;i++)
23.    p=p+add(x,0);
24.   return p;
25. }              /* end mul()   */
26. int div ( int x, int y)
27. {
28.   int p=0;
29.   while(x>=y)    /* while(x>y) is the original statement */
30. {
        a.   x=sub(x,y);    b.   p=p+1;
31. }
32. return p;
33.           }          /*  end div()  */
```

**Fig. 2.** C language Program for Basic Mathematical Operations

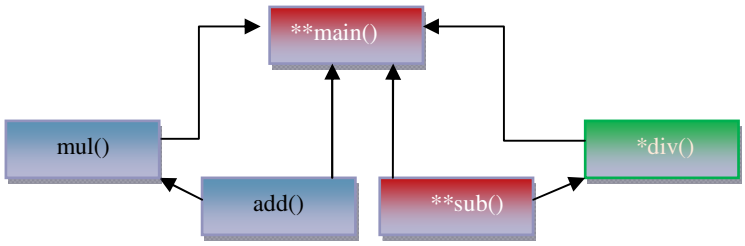


**Fig. 3.** Call Graph of Program (Fig. 2)

The assumed test case coverage of modules of Fig. 3 is shown in Table 1. From Table 1, it can be derived from Table 1 that test suite required for the execution of changed module and affected modules will be consist of $T_2$, $T_3$, T4, $T_5$, and $T_6$. In this

example, TCM array will store values {div(), sub(), main()}, TS will store three test suites corresponding to each module present in TCM i.e. {{ $T_4$,$T_6$},{ $T_2$, $T_6$},{ $T_3$,$T_4$, $T_5$ }}. Time array will store corresponding values of execution time of respective test suites i.e. Time {3, 4, 10}.

**Table 1.** Test Coverage Representation of Modules (Fig. 3)

| Module | Test cases required for coverage | Time taken(in seconds) |
|---|---|---|
| main() | $T_3$,$T_4$, $T_5$ | 10 |
| add() | $T_1$,$T_5$ | 8 |
| sub() | $T_2$, $T_6$ | 4 |
| mul() | $T_3$,$T_5$ | 9 |
| div() | $T_4$,$T_6$ | 3 |

TIM for these modules can be calculated by using Equations (1), (2), and (3). HTI value has been considered as 20, and both modules sub () and div () are called 5 times in 20 executions of program. TIM values of the required modules have been tabulated in Table 2. From TIM values, it can be concluded that module main () is the most important and module sub () has the least importance. Thus, TCM will have modules arranged in order {main (), div (), sub ()}. Accordingly, TS will also get rearranged and the element will be {{T3, T4, $T_5$}, {$T_4$, $T_6$}, {T2, $T_6$}}. Time array will have vales {10, 3, 4}. The prioritized test set P for this example will contain test cases in order $T_3$, $T_4$, $T_5$, $T_6$, and $T_2$ according to the importance of modules of estimated impact set.

**Table 2.** TIM Values of Changed/Affected Modules

| Module | UIM | DIM | THC(Module) | HVM | TIM |
|---|---|---|---|---|---|
| main() | 4 | 0.333333 | 20 | 20 | 26.66667 |
| sub() | 2 | 0.166667 | 5 | 5 | 1.666667 |
| div() | 3 | 0.166667 | 5 | 5 | 2.5 |

According to the proposed algorithm represented in Fig. 1, depending upon the value of time budget available for regression testing, all test cases for selected module/modules will be executed. For example, if time budget is 10, only module main () will be tested.

### 5.1 Major Findings

For the evaluation of the proposed algorithm (Fig. 1) a small C language program (Fig. 2) , has been considered. From the results of this experiment, it has been observed that the importance of a module plays a significant role in prioritizing a test suite for regression testing. UIM, DIM and HVM metrics proposed in this paper facilitates in the calculation of TIM of a module. Even, if the time budget allowed is greater than the time required to execute the prioritized test suite, prioritization of test cases is important for additional coverage of modules. Further, it has been observed that there is the least possibility for any two modules to have the same value of TIM. This will help to reduce the random selection of test cases at the time of prioritization.

### 5.2 Threats to Validity

Main threat to validity is that two main contributors towards TIM, i.e. UIM and HVM are decided by user and developer respectively. Though, the metrics, TIM, UIM, DIM, and HVM are effectual to find the importance of a module, still, Generalization of the results of proposed algorithm requires more empirical studies. The correlation of metrics UIM, DIM, and HVM used in the algorithm to generate TIM, needs extensive investigational studies for its universal acceptance.

## 6 Conclusions

In prioritization of test cases for regression testing, the role of a module is very significant. The modules of *TCM* set should be tested according to the order of their importance. *TIM* algorithm, present in the current study, prioritizes test cases on the basis of the importance of their respective modules. The metrics proposed in this study played an imperative role in finding the importance of modules. The current study can be extended further by applying *TIM* algorithm to programs of different sizes, types, and complexities.

## References

1. Breech, B., Danalis, A., Shindo, S., Pollock, L.: Online impact analysis via dynamic compilation technology. In: ICSM 2004, September 11-14 (2004)
2. Chechik, M., Winnie, L., Nejati, S., Cabot, J., Diskin, Z., Eaterbrook, S., Sabetzadeh, M., Salay, R.: Relationship-based change propagation: a case study. In: MiSE 2009, May 17-18 (2009)
3. English, M., Exton, C., Rigon, I.: Fault detection and prediction in an open-source Software project. ACM, New York (2009)
4. Engstrom, E., Skoglund, M., Runeson, P.: Empirical evaluation of regression test selection techniques: a systematic review. In: ESEM 2008, October 9-10 (2008)
5. German, D.M., Robles, G., Hassan, R.E.: Change impact graph: determining the impact of prior code changes. In: IEEE Working Conference on Source Code Analysis and Manipulation, SCAM (2008)
6. Kagdi, H.Z., Jonathan, I.M.: Software-change prediction: estimated+ actual. In: Proc. IEEE Workshop on Software Evolvability, pp. 38–43 (2006)
7. Ma, Z., Zhao, J.: Test case prioritization based on analysis of program structure. In: APSEC (2008)
8. Maia, C.L.B., Refael, A.F.D.C., Fabricio, G.D.F.: Automated test case prioritization with reactive GRASP. In: Advances in Software Engineering. Hindawi Publishing Corporation (2010)
9. Nadi, S., Holt, R., Davis, I., Mankovskii., S.: DRACA: decision support for root cause analysis and change impact analysis for CMDBs (2009)
10. Orso, A., Apiwattanapong, T., Law, J., Rothermel, G., Harrold, M.J.: Leveraging field data for impact analysis and regression testing. In: ESEC/FSE 2003, September 1-5 (2003)
11. Orso, A., Apiwattanapong, T., Law, J., Rothermel, G., Harrold, M.J.: An empirical comparison of dynamic impact analysis algorithms. In: ICSE 2004 (2004)

12. Park, S., Kim, H., Bae, D.H.: Change Impact analysis of a software process using process slicing. In: QSIC (2009)
13. Rajlich, V., Patrenko, M.: Variable granularity for improving precision of impact analysis. In: ICPC (2009)
14. Rao, A.A., Reddy, K.N.: Detecting bad smells in object oriented design using design Change propagation probability matrix. In: IMECS 2008, Hong Kong (March 2008)
15. Ryder, B.G., Ren, X., Shah, F., Tip, F., Chesley, O., Chianti, J.: A tool for change Impact analysis of java program. In: OOPSLA 2004, October 24-28 (2004)
16. Sanjeev, A.S.M., Wibowo, B.: Regression test selection based on version changes of components. In: Proceedings of APSEC 2003 (2003)
17. Sherriff, M., Lake, M., Williams, L.: Prioritization of regression tests using singular value decomposition with empirical change records. In: International Symposium on Software Reliability Engineering (November 2007)
18. Sherriff, M., Williams, L.: Empirical software change impact analysis using singular value decomposition. In: ICST (2008)
19. Tie, F., Maletic, J.I.: Applying dynamic change impact analysis in component-based Architecture design. In: ACIS International Conference on Software Engineering (2006)
20. Walcott, K.R., Kapfhammer, G.M., Soffa, M.L., Roos, R.S.: Time-aware test suite prioritization. In: Proceedings of ISSTA 2006, July 17-20 (2006)
21. Yoo, S., Harman, M.: Regression testing minimization, selection, and prioritization: a survey. Software Test. Verif. Reliab. (2007)
22. Yu, Z., Rajlich, V.: Hidden dependencies in program comprehension and change Propagation. In: IWPC (2001)