Journal of Information Processing Systems, Vol.6, No.4, December 2010 DOI : 10.3745/JIPS.2010.6.4.597

A Dynamic Approach to Estimate Change Impact using Type of Change Propagation

Chetna Gupta*, Yogesh Singh** and Durg Singh Chauhan***

Abstract—Software evolution is an ongoing process carried out with the aim of extending base applications either for adding new functionalities or for adapting software to changing environments. This brings about the need for estimating and determining the overall impact of changes to a software system. In the last few decades many such change/impact analysis techniques have been developed to identify consequences of making changes to software systems. In this paper we propose a new approach of estimating change/impact analysis by classifying change based on type of change classification e.g. (a) nature and (b) extent of change propagation. The impact set produced consists of two dimensions of information: (a) statements affected by change propagation and (b) percentage i.e. statements affected in each category and involving the overall system. We also propose an algorithm for classifying the type of change. To establish confidence in effectiveness and efficiency we illustrate this technique with the help of an example. Results of our analysis are promising towards achieving the aim of the proposed endeavor to enhance change classification. The proposed dynamic technique for estimating impact sets and their percentage of impact will help software maintainers in performing selective regression testing by analyzing impact sets regarding the nature of change and change dependency.

Keywords—Change Impact Analysis, Regression Testing, Software Maintenance, Software Testing

1. Introduction

Software evolution is an on-going process which moves towards enhancement of existing software systems, involving both development and maintenance. Software systems are mainly changed due to new requirements and technological changes which often lead to modification of software systems. Studies indicate that 90% [1] of all software development is maintenance and more than 50% of the total maintenance cost of software lies in rework i.e. in changing the software [2, 3].

Change management, impact analysis (IA) and regression testing are the three main steps in the maintenance process. The software maintenance process can only be optimized if precise and unambiguous information is available about the potential ripple effects of a change to an existing system. Making changes to software without an understanding and knowledge of soft-

Manuscript received July 15, 2010; first revision November 4, 2010; accepted November 9, 2010.
Corresponding Author: Chetna Gupta
* Dept. of Computer Science and Engineering/ Information Technology, Jaypee Institute of Information Technology, Noida, India (chetnagupta04 @gmail.com)
** University School of Information Technology, Guru Gobind Singh Indraprastha, University, Kashmere gate, Delhi, India (ys66@rediffmail.com)
*** Uttarakhand Technical University, Dehradun, India (pdschauhan@gmail.com)

ware components can produce disastrous effects [4] and can lead to degraded software. As a result the need arises for software maintainers to estimate the kinds of impact a change may cause. However, identification of these changes is not an easy task as any modification in one part of software may have subsequent ripple effects on other related components within the software. Thus calculating ripple effects before or after making changes is valuable in identifying the potential impact of a possible change. If done in a proactive manner i.e. before the changes are made it can be helpful in predicting the effects of the proposed changes in terms of their affect on the overall system and the corresponding costs while simultaneously providing the maintainer the option to select various alternatives. On the other hand, if applied after modifications, it can help in reducing the risks associated with releasing modified software by alerting engineers to potentially affected program components. Software change impact analysis, often called simply impact analysis, is a family of approaches for addressing these problems [4-8]

Impact Analysis is defined in the IEEE Standard for Software Maintenance [9] as "*impact analysis: Identifies all system and software products that a change request affects and develops an estimate of the resources needed to accomplish the change. This includes determining the scope of the changes to plan and implement work, accurately estimating the resources needed to perform the work, and analyzing the requested change's cost and benefits.*"

Although impact analysis has its natural place in requirement engineering as changes to software often are initiated by changes in requirements, ironically the research related to impact analysis is more commonly found in software maintenance literature. IA can also be viewed as an integral part of every phase of software development; for instance during requirements development, design, coding and implementation. Any demand or requirement for new and various requirements have an effect on existing requirements which leads to changes in design and code.

The primary goal of impact analysis is to identify software objects affected (or possibly affected) by proposed changes. With knowledge of these identified affected objects the software maintainer can construct resource estimates that can be used to guide maintenance activities. Impact analysis has three primary benefits for a software maintainer: improved accuracy of resource estimates, and thus better maintenance scheduling and reduced change costs; a reduction in the amount of corrective maintenance because of fewer introduced errors and improved software quality [10].

In this paper, we propose a new approach for estimating change impact analysis (CIA) based on change classification, which will not only estimate the impact of change to an overall system but at the same time will provide information regarding the nature of the change(s) and the corresponding affected set of statements and their percentage of impact. The software maintainer can then use this information to perform selective regression testing. The main contribution of this work is as follows:

- A novel change classification system to classify run time data into four categories in order to determine the nature of a change or changes and to measure the impact propagation on a software system
- An algorithm to categorize and to collect impact sets based on classification.

These contributions are helpful in estimating the impact of such changes thereby aiding in selecting useful test cases for regression testing.

The remainder of this paper is organized as follows: In the next section, related work is presented. We describe our proposed classification system, framework and algorithm in section 3. In section 4 we present some experimental results of our proposed scheme, section 5 presents application of the proposed work and finally section 6 presents the conclusion.

2. Related Work

When planning modifications, impact analysis helps maintainers to predict effects and costs of planned changes. These Impact Analysis techniques are broadly classified into (1) Static impact analysis techniques and (2) Dynamic impact analysis techniques. The next section provides a brief overview of these two categories.

2.1 Brief Overview of Static Impact Analysis

Static impact analysis is a technique used to analyze a software system without actually executing the source code. In most cases static analysis is performed on some version of source code and in other cases some form of object code. Decades of research have shown that static analysis based techniques can safely estimate the impact of changes but their conservative assumptions often result in impact sets that include most of the software. In some cases the impact set produced includes more than 90% of the program [11]. Such impact sets make the results of the impact analysis almost useless for other software-engineering tasks. For example, regression-testing techniques that use impact analysis to identify which parts of the program to retest after a change would have to retest most of the program. Static IA techniques listed in literature are further partitioned into two types: (a) traceability analysis and (b) dependency analysis. Traceability analysis techniques calculate static impact set by tracing the software development cycle from software requirements through design, code and testing. Dependency based analysis techniques [2, 5-7, 12-19] works on the principle of analyzing program syntax for semantic dependencies among program entities such as linkages between parts, variables, logic, modules, thereafter it calculates impact sets according to dependencies. A brief summary of a comparison between these techniques is provided in [20].

As stated above, a static impact analysis requires access to source code and thus the calculation of impact set is based on a static program profile synthesized with assumptions of possible system behaviors. Hence, this type of analysis technique is either imprecise, unsafe, or both, and tends to overestimate the effects of changes. Thus, the results of a static IA are hardly useful for software engineering tasks.

The problem with static analysis based approaches can be divided into two. Firstly, they consider all possible behaviors of the software, whereas, in practice, only a subset of such behaviors may be exercised by the users. Secondly, and more importantly, they also consider some impossible behaviors, due to the imprecision of the analysis. Therefore, recently, researchers have investigated and defined impact-analysis techniques that rely on dynamic, rather than static, information about program behavior [20-22]

2.2 Brief Overview of Dynamic Impact Analysis

The main aim of dynamic impact analysis is to analyze a software system by gathering program behaviors attained by executing programs on a real or virtual processor. It gathers impact sets by analyzing program behaviors for a specific set of executions (at least one of the considered program executions) during run time (i.e. data is obtained from executing a program). Dynamic IA does not require access to source code or linking process. Instrumentation and calculation of dynamic IA cause overhead in both time and space and results produced are much more precise than those produced by static impact analysis techniques. Various techniques mentioned in literature are based on collecting impact sets of dynamic program behavior. [20] Performs at the method level, based on whole path profiling [23]. It produces traces of procedure names, function returns and program exits in the order in which they occur in multiple executions. [22] Is an extension of [20] that allows an algorithm to collect data incrementally so that it becomes cost effective to re-compute (i.e. re-collect) data required for newer versions of the software. [11] Uses light weight instrumentation and collects coverage information of methods per executions. It also works at the method level, but uses coverage, rather than trace, information to compute impact sets. The coverage information for each execution is stored in a bit vector that contains one bit per method in the program. The comparison of [20] and [11] is given in [24]. The generic approach of [25] is based on an EA (execute after) relation for efficiently collecting and analyzing collected information dynamically. This method is based on identifying all program entities that are executed after *e*, where *e* is the set of executions for some procedure *p* in the considered program execution. Therefore, it computes a binary relation for each pair of entities *e1* and *e2* in P, where *e2* is executed after *e1* in any of executions in E. In other words, it finds all those methods that are executed after changed methods. The impact set produced is thereby a union of methods executed after any changed methods in all executions considered. [26] Calculates impact sets by finding methods that are executed after methods in a changed set and includes them in the impact set. [27] Proposes a methodology for determining the impact of new system modification(s) by analyzing software change records through singular value decomposition. It generates clusters of files that historically tend to change together to address faults and failures found in code base. [28] Proposes a novel change impact analysis method based on the idea of mutual relationships between software objects that can be inferred using a statistical learning approach.

On the other hand if applied online, it can calculate impact sets concurrently with program execution [29]. Online dynamic impact analysis has the same goal as dynamic impact analysis, but online impact analysis is performed concurrently with program execution rather than calculating impact sets from executing a program. [30] Presents a technique based on analysis of program dependency in terms of variable definition and usage to generate impact sets. [31] Provides framework for collecting impact traces completely online and it also provides support for impact visualization for regression testing.

The focus of this research is to gather data for impact sets dynamically i.e. on executing the program as compared to static impact analysis techniques. Unlike other techniques it computes impact sets by tracing change and then classifying it into proposed categories to help software maintainers understand the nature (type) of change and its impact and propagation. It will also assist in selecting only those test cases which fall into those categories for regression testing.

3. BASIC METHOD AND PROPOSED APPROACH

This section describes our approach for classifying the type of change and its impact on (introduction of any change in) software. This technique takes into account both change (type) and its ripple effects on the system by storing change propagation during execution. The impact sets are generated in terms of affected statements and percentage of change impact. To discuss the presented approach in detail we first explain the proposed classification system adopted for our framework followed by discussion of the framework itself. Next we explain the algorithm which implements the technique followed by data collection and result observation of the overall technique.

3.1 Proposed Classification System

We propose a classification system that classifies the change (Δx) of any form (i.e. changes made due to new requirements or technological changes) made to software systems into the following four categories.

(a) Functional Change Impact: It is defined as any functional change (Δf) in some function x (fun (x)) by adding, deleting or modifying statements may directly or indirectly propagate its impact to other related functions of a program. These changes can therefore be classified as functional changes.

(b) Logical Change Impact: It is defined as any change (addition, deletion or modification) that corresponds to a change in logic or decision in program code. It is based on a representation of control flow graph (CFG) and therefore any logical change (Δl) in the decision flow of the control flow graph implies that it can propagate its affect in the system accordingly. A CFG is a graph notation (diagrammatic representation) of a program and its execution which represents all possible sequences of statements of a program. It is a directed graph with vertices and edges where vertices are statements and edges represent flow of control. It consists of a start node, an end node and flows (or arcs) between nodes. Each node is labeled in order for it to be identified and associated correctly with its corresponding part in program code. The nodes are either entire statements or fragments of a statement and edges represent flow of control. If *i* and *j* are nodes in the program graph, there is an edge from node *i* to node *j* if the statement (fragment) corresponding to node *j* can be executed immediately after the statement (fragment) corresponding to node *i* [32]. Logical change impact connects mostly intra-components which can lead to functional and behavioral change also.

(c) Structural Change Impact: It is defined as any structural modification in a software system: (a) an addition of a new piece of code in an already existing program or the combining of two programs into one (b) deletion of some already existing code due to splitting of a program into two programs or deletion of a piece of code from a software program.

(d) Behavioral Change Impact: It is defined as any change in behavior of a program (Δb) component that can cause a potential change in behavior of related components. A behavior of a program describes the execution order of activities including decision making, entry and exit criteria of a program. The aim of estimating this type of change is to simply make sure that the output complies with the requests of the system.

3.2 Proposed Framework for Change Impact Analysis

The framework presented firsts highlights changed areas by comparing two source codes line by line and later uses classification results to estimate impact of changes thereby calculating its

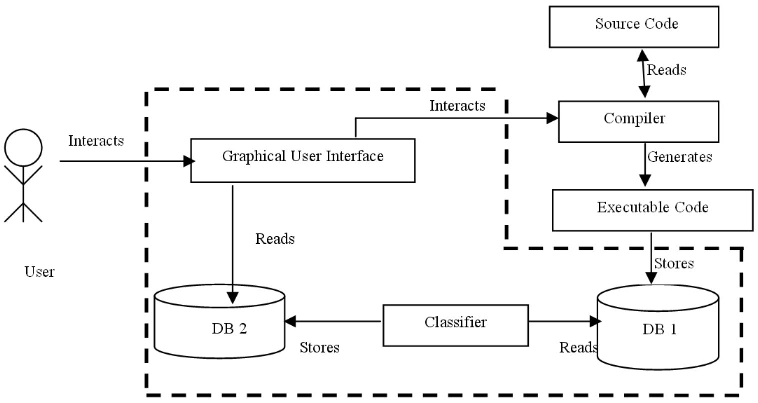


Fig. 1. Framework for Visualization of Change Impact Analysis

percentage of impact to pin point irregularities in a straightforward manner. The whole process is explained with its main components (shown in bold dotted line) in Fig. 1.

(a) Graphical User Interface: is a user friendly interface to provide easy access to users for updating two versions of the program. It interacts with a compiler and stores results of the comparison of both programs in database 1.
(b) Database 1 (DB1): stores execution history and the control flow graph of two versions of a program for analyzing two versions of code.
(c) Classifier: reads the analyzer and classifies data (using an algorithm) according to the classifications provided in section 3.1. We use the following formulae to calculate the percentage of impact:

$$\text{Percentage of change} = \frac{\textit{Total number of statements where change has propagated}}{\textit{Total number of statements in a program}} \qquad (1)$$

(d) Database 2 (DB2): stores results of classification obtained from the classifier for analysis.

3.3 Algorithm

The algorithm presented in Fig. 2 classifies comparison information to generate dynamic impact sets. To calculate Functional Change Impacts we create a control flow graph of a program and analyze connectivity of changed statement(s) to other statements defined in other functions of a program. It adds all those statements which are changed and has a direct impact on other statements of related functions of a program to compute impact set. To calculate Logical Change Impacts it analyzes all those statements in a CFG where there is a change in logic or decision in program code. It adds all those statements whose logic has been changed and all those statements which are related to those statements in terms of dependency (either by definition or usage) to compute an impact set. To calculate Structural Change Impacts made to the code it finds all those statements which have been added or deleted to and from the original code. To calculate Behavioral Change Impacts it adds all those statements to impact sets where there is any change in behavior of the program from the original one.

```
//Reachable( ) would return TRUE if there may be a direct impact of current statement on a different
function
    for i=0; i< n // n = number of rows in the table generated
            if(sign[i] = = '+')
                        If (Reachable(Program2[i]))
                                    Δf = Δf ∪ Program2[i];
            else if(sign[i] = = '-')
                        If (Reachable(Program1[i]))
                                    Δf = Δf ∪ Program1[i];
    // Decision( ) would return TRUE if the statement changed contains a decision or a comparison
            for i=0; i< n // n = number of rows in the table generated
            if(sign[i] = = '+')
                        If (Decision(Program2[i]))
                                    Δl = Δl ∪ Program2[i];
            else if(sign[i] = = '-')
                        If (Decision(Program1[i]))
                                    Δl = Δl ∪ Program1[i];
    // isFunction( ) would return TRUE if the statement defines a new function
    for i=0 to i= n;            // n = total number of change highlighted in the Result of
                                       Comparison table stored in database
            if(sign[i] = = '+" &&   isFunction(Prorgam[2]))
                        Δs = Δs ∪ Program1[2]
            else if(sign[i] = = '-'&& isFunction (Prorgam[1]))
                        Δs = Δs ∪ Program2[1]
    for i=0; i< n // n = number of rows in the table generated
            if(sign[i] = = '+')
                                    Δb = Δb ∪ Program2[i];
            else if(sign[i] = = '-')
                                    Δb = Δb ∪ Program1[i];
```

Fig. 2. Algorithm

4. Data Collection and Result Observation

The following steps were followed for collection of relevant data in this study:

(a) There are two versions of each program: the original and the modified.
(b) Then the two source codes are uploaded in a graphical user interface for comparisons and their execution history (table of results of comparison) and CFG are stored in a database.
(c) We then apply the proposed technique, i.e. procedure to classify the nature of change and equation (1) to calculate percentage of impact.

```
1.  int sum(int *a, int s){
2.  int sm=0, i;
3.  for(i=0;i<s; i++)
4.  {
5.  if(a[i]<0)
6.  a[i]=-a[i];
7.  sm + = a[i];
8.  }
9.  return sm;
10. }
11. int avg(int *a, int s){
12. int av, sm;
13. sm = sum(a, s);
14. av = sm/s;
15. return av;
16. }
```

Fig. 3. Program 1

```
1.  int sum(int *a, int s){
2.  int sm=0, i;
3.  for(i=1;i<=s; i++)
4.  {
5.  sm + = a[i-1];
6.  }
7.  return sm;
8.  }
9.  int avg(int *a, int s){
10. int av, sm;
11. input(a, s);
12. sm = sum(a, s);
13. av = sm/s;
14. return av;
15. }
16. input(int *a, int s){
17. for(int i =0;i<s;i++)
18. scanf("%d", &a[i]);
19. }
```

Fig. 4. Program 2

This result will then be used by the software maintainer(s) to predict the impact of introducing a change. To validate the presented techniques and to assess the usefulness of classifying the type of change and calculation of percentage of change introduced for impact analysis and regression testing, we performed a set of empirical studies. We explain the whole process of collecting the traces with the help of the following example given in Fig. 3 and Fig. 4.

4.1 Database 1

It stores a table of comparison and corresponding CFG of two versions of code. It generates a table of comparison by comparing two source codes line by line to examine the differences between two pieces of code. The result of the comparison is depicted in Table 1 where in the sign column the equal to sign (=) means two lines are similar in program 1 and program 2 (b) the subtraction sign (-) means the line has either been deleted or it does not exist in other source code; the addition sign (+) means a new line of code has been added. In column 3 and 5 the (--) sign means statements are missing in the other program.

4.2 Classifier

Now the classifier will read Table 1 from the database and will classify the information using the proposed algorithm and four categories.

The algorithm will trace the change propagation and according to the defined categories it will classify change propagation for each category. The impact sets for each category consists of

Table 1. Result of comparison stored in DB1

Sign	Line No	Program 1	Line No	Program 2
=	1	int sum(int *a, int s){	1	int sum(int *a, int s){
=	2	int sm=0, I;	2	int sm=0, I;
-	3	for(i=0;i<s; i++)	--	--
-	4	{	--	--
-	5	if(a[i]<0)	--	--
-	6	a[i]=-a[i];	--	--
-	7	sm + = a[i];	--	--
+	--	--	3	for(i=1;i<=s; i++)
+	--	--	4	{
+	--	--	5	sm + = a[i-1];
=	8	}	6	}
-	9	return sm;	7	return sm;--
=	10	}	8	}
=	11	int avg(int *a, int s){	9	int avg(int *a, int s){
=	12	int av, sm;	10	int av, sm;
+			11	input(a, s);
=	13	sm = sum(a, s);	12	sm = sum(a, s);
=	14	av = sm/s;	13	av = sm/s;
=	15	return av;	14	return av;
+	--	--	15	}
+	--	--	16	input(int *a, int s) {
+	--	--	17	for(int I =0;i<s;i++)
+	--	--	18	scanf(“%d”, &a[i]);
+	--	--	19	}

Table 2. Classification results obtained by classifier

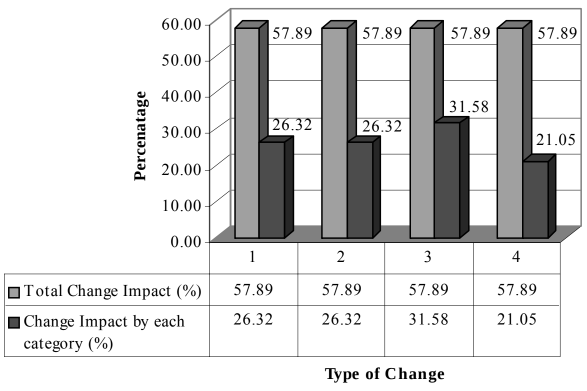
Type of change	Actual change	Statements where change has propagated	Impact of Change
Functional Change Impact	<deleted line 3, 4, 5, 6 in P1>	5, 7, 12, 13, 14	26.32%
Logical Change Impact	3	5, 7, 12, 13, 14	26.32%
Structural Change Impact	11, 16, 17, 18	3, 5, 7, 12, 13, 14	31.58%
Behavioral Change Impact	5	7, 12, 13, 14	21.05%

Table 3. Result analysis for overall system

	Affected Line No	Percentage of total change impact on overall system
Total statements affected	3, 5, 6, 7, 11, 12, 13, 14, 16, 17, 18	57.89%

a collection of affected nodes. For example, there is change in line number 3 of program 1, where $i<s$ is changed to $i<=s$. On tracing this change it can be seen that it has potential effect on lines number 5, 7, 12, 13 and 14 of program 2. According to the classification defined in section 3.1, it is clear that this change contributes to three different categories at the same time, namely functional change, logical change and structural change. Table 2 provides the list of affected statements for each category along with the percentage of change impact computed using formulae given in equation 1. To estimate impact of change on the overall system it summarizes the data in Table 3 which provides the list of affected statements for change propagation of the overall system.

Results of Table 2 and 3 can be used by the software maintainer(s) to estimate impact of change by analyzing the type of change, namely Functional Change Impact, Logical Change Impact, Structural Change Impact and Behavioral Change Impact and total number of statements in a program where change has propagated. Fig. 5 above represents analysis of impact of



	1	2	3	4
▨ Total Change Impact (%)	57.89	57.89	57.89	57.89
■ Change Impact by each category (%)	26.32	26.32	31.58	21.05

Fig. 5. Summary of percentage of impact of change: where 1 represents total change impact vs Functional Change Impact, 2 represents total change impact vs. Logical Change Impact, 3 represents total change impact vs. Structural Change Impact and 4 represents total change impact vs. Behavioral Change Impact

change on the overall system. It can be noted here that the sum percentage change of each the classification categories is more than 100%; this is because of overlapping statements (occurrence of more than one statement) in each category. The results of our experiment show that this technique can efficiently classify information into defined categories and hence can reduce the time and cost of software maintenance tasks. This will assist in selective regression testing as it will help software testers to choose only those test cases where the actual change and its impact has propagated from the old test suite to be executed on the modified version of software. It will serve in establishing confidence in modified programs thereby increasing levels of customer satisfaction. Also it will benefit in saving time and cost of software maintenance work as experiences have shown how a small change can adversely affect software if its impact is not understood properly.

5. Application of Proposed Work

Software practitioners may use the presented technique to analyze details of a particular impact propagated by an introduction of change to a software system. The proposed approach supports software practitioners by (a) providing knowledge of the nature of change(s) (b) providing knowledge of all other parts that are affected by a change propagation (c) providing measurement of impact on various parts of program. This technique may lead to greater savings of time and cost of maintenance labor. The application of this work may improve the quality, reliability, and effectiveness of the code, which may, in turn, increase the level of customer satisfaction.

6. Conclusion

In this paper we have presented a new approach for change impact analysis to predict the percentage impact of change through a change classification for software systems. The proposed technique first computes comparison results in two versions of program and then classifies the obtained results using the type of change classification proposed in section 3.1. The results are promising in achieving effectiveness and efficiency of the technique, by (a) classifying the nature of change (b) estimating impact percentage of change(s) to the overall software system (b) estimating impact sets in terms of affected statements and percentage of such changes on the overall system and (c) assisting software maintainers in performing selective regression testing by analyzing results produced by our classification algorithm.

Reference

[1] R. C. Seacord, D. Plakosh, G. A. Lewis, "Modernizing Legacy Systems: Modernizing Legacy Systems: Software Technologies, Engineering Processes, and Business Practices," Addison-Wesley, 2003.

[2] M. Lee, A. J. Offutt, R.T. Alexander, "Algorithmic Analysis of the Impacts of Changes to Object-oriented Software," The Technology of Object-Oriented Languages and Systems, 2000, pp.61.

[3] G. J. Myres, "Art of Software Testing," John Wiley & Sons, New York, 1979.

[4] S. Bohner and R. Arnold, "Software Change Impact Analysis," Proceedings of IEEE Computer Society Press, Los Alamitos, CA, USA, 1996.

[5] R. S. Arnold, S. A. Bohner, "Impact analysis - towards a framework for comparison," Proceedings of IEEE International Conference on Software Maintenance, Montreal, Que, Can, September, 1993,

pp.292-301.
[6] J. P. Loyall, S. A. Mathisen, C. P. Satterthwaite, “Impact analysis and change management for avionics software,” Proceedings of IEEE National Aerospace and ElectronicsConference, Part 2, Dayton, OH, July, 1997, pp.740-747.
[7] S. L. Pfleeger, “Software Engineering: Theory and Practice,” Prentice Hall, Englewood Cliffs, NJ, 1998.
[8] R. J. Turver, M. Munro, “Early impact analysis technique for software maintenance,” Journal of Software Maintenance: Research and Practice, 6(1):35-52, January, 1994.
[9] International Standard - ISO/IEC 14764 IEEE Std 14764-2006, IEEE Standard for Software Maintenance, IEEE Computer Society.
[10] R. Moreton, “A process model for software maintenance,” Journal of Information Technology, 5:100-104, 1990.
[11] Orso, T. Apiwattanapong, M. J. Harrold, “Leveraging field data for impact analysis and regression testing”, Proceedings of the ACM SIGSOFT Symposium on Foundations of Software Engineering, September, 2003, pp.128-137.
[12] B. G. Ryder, F. Tip, “Change impact analysis for object oriented programs”, Proceedings of the ACM Workshop on Program Analysis for Software Tools and Engineering, October, 2001, pp.46-53.
[13] L. C. Briand, Y. Labiche, L. O’Sullivan, “Impact Analysis and Change Management of UML Models”, Proceedings of the International Conference on Software Maintenance (ICSM’03), 2003.
[14] M.Weiser, “Program slicing”, Proceedings of 5th IEEE International Conference on Software Engineering, San Diego, CA, March, 1981, pp.439-49.
[15] H. Agrawal, J. Horgan, “Dynamic program slicing”, Proceedings of SIGPLAN ’90 Conference on Programming Language Design and Implementation. SIGPLAN Notices., White Plains, June, 1990, ACM, pp.246-56.
[16] S. Horwitz, T. Reps, D. Binkley., “Interprocedural Slicing Using Dependence Graphs”, ACM Trans. Prog. Lang. Syst., Vol.12(1), January, 1990, pp.27-60.
[17] M. Kamkar, “An Overview and Comparative Classification of Program Slicing Techniques”, Journal of Systems Software, Vol.31(3), 1995, pp.197-214.
[18] B. Korel, J. Laski, “Dynamic slicing in computer programs”, Journal of Systems Software, Vol.13(3), 1990, pp.187-95.
[19] L. Li, A. J. Offutt, “Algorithmic analysis of the impact of changes to object-oriented software”, Proceedings of IEEE International Conference on Software Maintenance, Monterey, CA, USA, November, 1996, pp.171-184.
[20] J. Law, G. Rothermel, “Whole program path-based dynamic impact analysis”, Proceedings of the International Conference on Software Engineering, May, 2003, pp.308-318.
[21] B. Breech, A. Danalis, S. Shindo, L. Pollock., “Online impact analysis via dynamic compilation Technology”, Proceedings of the International Conference of Software Maintenance, September, 2004.
[22] J. Law, G. Rothermel, “Incremental dynamic impact analysis for evolving software systems”, Proceedings of the International Symposium on Software Reliability Engineering, November, 2003.
[23] J. Larus. Whole Program Paths. In Proc. SIGPLAN PLDI 99, Atlanta, GA, May, 1999. ACM, pp.1-11.
[24] A. Orso, T. Apiwattanapong, J. Law, G. Rothermel, M. J. Harrold, “An empirical comparison of dynamic impact analysis algorithms”, Proceedings of the International Conference on Software Engineering, May, 2004, pp.491-500.
[25] T. Apiwattanapong, A. Orso, M. J. Harrold, “Efficient and Precise Dynamic Impact analysis using Execute-After Sequences”, Proceeding of ACM- International Conference on Software Engineering (ICSE), St. Louis, Missouri, USA, 2005.
[26] L. Huang, Dr. Y.T. Song, “Dynamic Impact Analysis Using Execution Profile Tracing”, Proceedings of the Fourth International Conference on Software Engineering Research, Management and Applications (SERA), 2006.
[27] M. Sherriff, L. Williams, “Empirical Software Change Impact Analysis using Singular Value Decomposition”, Proceedings of 1st IEEE International Conference on Software Testing, Verification, and Validation (ICST), 2008.
[28] M. Ceccarelli, L. Cerulo, G. Canfora, M. D. Penta "An Eclectic Approach for Change Impact Analysis,” Proceedings of International Conference on Software Engineering (ICSE), 2010.
[29] B. Breech, M. Tegtmeyer, L. Pollock, "A Comparison of Online and Dynamic Impact Analysis Algorithms,” Proceedings in Ninth European Conference on Software Maintenance and Reengineering (CSMR'05), 2005.

[30] C. Gupta, Y. Singh, D. S. Chauhan, "An Efficient Dynamic Impact Analysis using Definition and Usage Information", International Journal of Digital Content Technology and its Applications, Vol.3 (4), 2009, pp.112-115.
[31] C. Gupta, Y. Singh, D. S. Chauhan, "DU-Regs: Online Dynamic Approach to Visualize Impact Analysis for Regression Testing", International Journal of Computer Applications, Vol.1(19), 2010, pp.8-11.
[32] K.K. Aggarwal, Y. Singh, "Software engineering," Third edition, New Age International Publishers, New Delhi, 2008.

Chetna Gupta
She is a Senior lecturer at Jaypee Institute of Information Technology, India. She holds a Masters of Technology and a Bachelor of Engineering degree in Computer Science and Engineering. Her areas of interest are Software Engineering, Requirement Engineering, Software Testing, Software Project Management, Data Structures and Web Applications. She has many publications in international journals and conferences to her credit. Currently she is pursuing her Ph.D. in Software Testing.

Yogesh Singh
He received his master's degree and Ph.D. degree in Computer Engineering from National Institute of Technology, Kurukeshtra, India. He is a professor in University School of Information Technology (USIT), Guru Gobind Singh Indraprastha University, Delhi, India. His research interests include software engineering focusing on the area of Software project planning, Testing, Metrics, Data Structures, Computer Architecture, Parallel Processing and Neural Networks. He is also a Controller of Examinations with the Guru Gobind Singh Indraprastha University. He was founder Head and dean of the University School of Information Technology of Guru Gobind Singh Indraprastha University. He is co-author of a book on software engineering, and is a Fellow of IETE and member of IEEE. He has more than 200 publications in international and national journals and conferences.

Durg Singh Chauhan
He received a Ph.D. degree from Indian Institute of Technology (IIT) Delhi in 1986, India and did his post doctoral work at Goddard space Flight Centre, Greenbelf Maryland. USA (1988-91). He is a Vice-Chancellor of Uttarakhand Technical University, Dehradun, India. Prior to this he also served as Vice-Chancellor in three other universities in India. He has been a member of the University Grant Commission (UGC), National Board of Accreditation (NBA) - executive, All India Council for Technical Education (AICTE), Council for Advancement of People's Action and Rural Technology (CAPART), National Accreditation Board for Testing and Calibration Laboratories (NABL) - Department of Science and Technology (DST) executive and member, National expert Committee for IIT- (National Institute of Technology) NIT research grants. He has authored two books and published and presented more than 115 research papers in international journals and international conferences and wrote more than 20 articles on various topics in national magazines. He has delivered hundreds of lectures in U.S. and Canadian universities.