DU-Regs: Online Dynamic Approach to Visualize Impact Analysis for Regression Testing

Chetna Gupta
Jaypee University of Information Technology
Waknaghat, District Solan
Himachal Pradesh, India

Yogesh Singh
University School of Information Technology
Guru Gobind Singh Indraprastha University
Kashmere gate, Delhi, India

Durg Singh Chauhan
Uttarakhand Technical Universirty
Dehradun, India

Maneesha Srivastav
Jaypee University of Information Technology
Waknaghat, District Solan
Himachal Pradesh, India

ABSTRACT
Software evolution is an ongoing process carried out by software maintainer's in order to meet the increasing demand, pressure and requirements for extending base applications for adding new functionalities, for fixing bugs or for adapting software to the changing environments. As a result, it establishes the need for estimating and determining the impact of changes on the overall software system. Impact Analysis is a way to estimate the impact of such changes either before or after the change is made. In the last few decades many such techniques and tools (both static and dynamic) have been proposed. In this paper we propose a new online dynamic impact analysis technique called Definition Usage-Regression Test Selection (DU-Regs), which collects impact traces completely online i.e. during execution. It works at statement level rather than on method level to capture more precise impact sets and at the same time, provides the support for impact visualization for regression testing.

Categories and Subject Descriptors
K.6.3 [**Software Management**]: Software Maintenance

General Terms
Verification

Keywords
Dynamic impact analysis, regression testing, software maintenance.

1. INTRODUCTION
Software systems undergo many changes because of changing requirements and pressure for extending base applications. More than 50% of the total maintenance cost of the software lies in the rework i.e. in changing the software [11, 4]. Making changes to the software without understanding and knowledge of the software component can produce disastrous effects [14] and can lead to degraded software. Impact analysis is the set of such techniques which are used to calculate or estimate the effects of the change on overall software system and addresses these problems (in terms of estimating the effect of the changes) [13, 8, 15, 12]. Impact Analysis can be applied before or after the changes are made. 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 the affect on the overall system and its corresponding cost and at the same time provides an option to the maintainer to select among various alternatives. On the other hand if applied after modifications, it can help in reducing the risks associated with releasing changed software by alerting engineers to potentially affected program components.

In this paper we propose a new online impact analysis technique, called DU-Regs, which is capable of collecting dynamic impact sets online. It is an online tool that captures traces while execution. This technique works at the statement level rather than on method level to produce more precise impact sets. The tool also provides the support to visualize impact sets through graphical interface.

2. RECENT WORK
Dynamic impact analysis techniques [7, 5, 2, 16, 1, 9] are based on dynamic program behaviors gathered for a specific set of

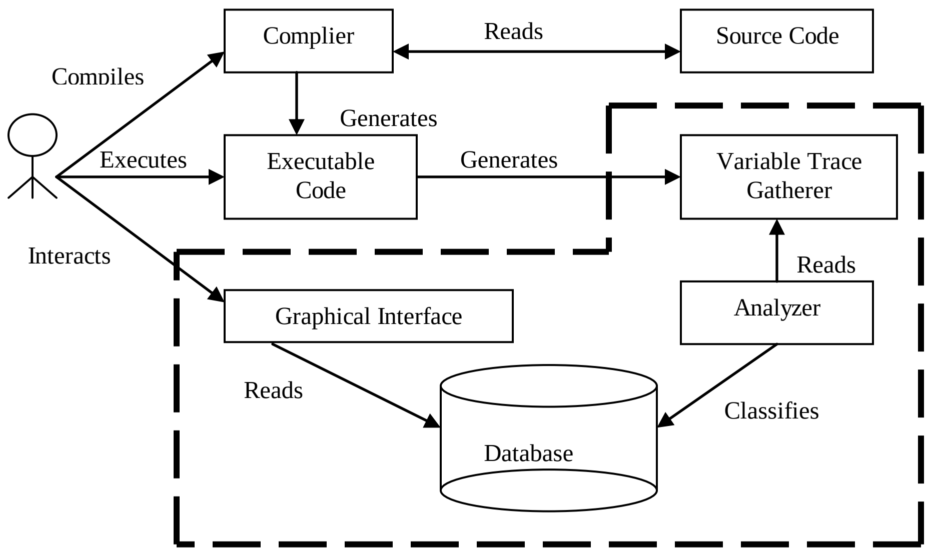

Figure 1. Model for DU-Regs.

executions (i.e. data obtained from executing a program) to perform analysis. PathImpact [7] works at the method level, based on whole path profiling [6]. It produces traces of procedure names, function returns and program exit in the order in which they occur in multiple executions. EvolveImpact [5] is the extension of the PathImpact [7] that allows PathImpact to collect data incrementally. CoverageImpact [2] uses light weight instrumentation and collect coverage information of methods per executions. It also works at the method level, but uses coverage, rather than trace, information to compute impact sets. CollectEA [16] is based on Execute After (EA) relation for efficiently collecting and analyzing the collected information dynamically. It identifies all program entities that are executed after e, where e is the set of executions for some procedure p in the considered program execution. It finds all those methods that are executed after the changed methods. A comparison of [7] and [2] is given in [1]. SVD-based impact analysis [9] determines the impact by analyzing software change records through singular value decomposition thereby generating clusters of files that historically tend to address faults and failures found in the code base.

On the other hand if applied online, it can calculate the impact sets concurrently with program execution [10].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 the impact sets from executing the program. Dynamic impact analysis does not require access to the source code or the linking process. Instrumentation and calculation of dynamic impact analysis cause overhead in both time and space. The result of the studies [3] indicates that performing impact analysis online can be more scalable than the dynamic counterparts.

3. OUR APPROACH

Our approach is summarized in Figure 1. The main components (shown in bold dotted line) along with their implementation issues are explained below:

- **Variable Trace Gatherer (VTG)**: to gather traces of every variable statement by statement on execution i.e. at run time.
- **Analyzer**: to read the data gathered by variable trace gatherer for classification. The classification is based on the definition and usage information of the variables used in the program. The defined classification is as follows:
    - d-def: definite definition
    - d-use: definite usage
    - p-def: predicate definition
    - p-use: predicate usage
- **Database**: stores the dynamic traces of the variables at run time for the above given classification
- **Graphical Interface**: reads the database to generate the affected statements either directly (on the basis of d-def and d-use) or indirectly (on the basis of p-def and p-use). The aim of using graphical interface is to provide insight and understanding to pin point irregularities.

As the model is completely based on online collection of dynamic traces, no access to the source is required for executing the program. Dynamic traces will be stored in database during run time. To permit visualization, the user can interact with the graphical interface, which in turn reads database.

```
         main () {
         int *p;
         int j, sum1, sum2;
1.       sum1 =0;
2.       sum2=0;
3.       read i, j;
4.       while (i<10) {
5.       if (j<0) {
6.       p =&sum1;
         } else {
7.       p = &sum2;
         }
8.       *p = add(j,*p);
9.       read j;
         }
10.      sum1 = add(j, sum1);
11.      print sum1, sum2;
         }
```

Figure 2. Example Program.

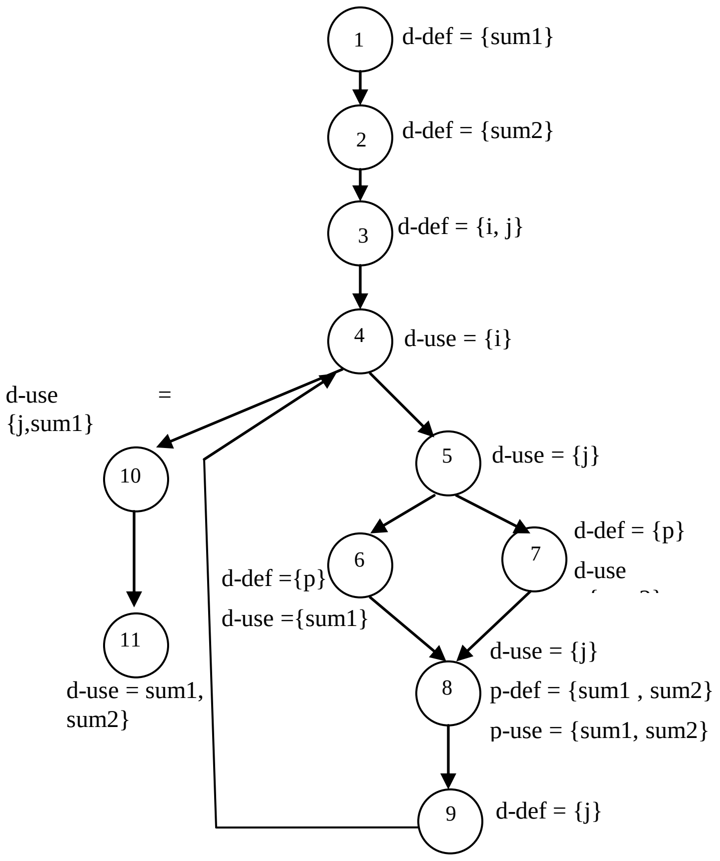

Figure 3. Control Flow Graph.

3.1 Example

To validate the presented techniques and to assess the usefulness of using defination and usage information of a variable 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 following example. Consider an example program in Figure 2. Suppose the change has occurred in statement 2 where, sum2 =0; is written as sum2 = 5; the corresponding control flow graph is shown in Figure 3.

3.1.1 Data collected by Variable Trace Gatherer

It will collect variable information as they appear in the program along with their statement numbers during run time. Table 1 provides the data collected by VTG.

Table 1. Data collected by VTG

Variables	Statement where they appear
sum 1	1, 6, 10, 11
sum 2	2, 7, 11
i	3, 4
j	3, 5, 8, 9, 10
p	6, 7, 8

3.1.2 Data collected by Analyzer

It will read variable trace gatherer and will store dynamic traces of each variable in databse after classifing it into above mentioned four catergories. Table 2 provides the classification results.

Table 2. Data collected by Analyzer

Variable	d-def	d-use	p-def	p-use
sum 1	1, 10	6, 10, 11	8	--
sum 2	2	7, 11	8	--
i	3	4, 10	--	--
j	3, 9	5, 8, 10	--	--
p	6, 7	--	--	--

3.1.3 Graphic Interface

It will read the database to generate dynamic impact set for analysis of software maintainers to underatnd and estimate the impact of change on overall system. Table 3 provides the results of suspicious statements where the change has propagated. Hence the results obtained by DU-Regs can be can be used for selective regression testing.

Table 3. Impact of change on overall system

Changed variable	Set of statements where the change has propogated
sum 2	2, 7, 8, 11

4. CONCLUSION

In this paper we have presented our new technique for calculating dynamic impact sets online at statement level. The technique classifies a data dependence based on the type of definition and usage which helps in analyzing the effect of the change on the overall system. This approach can prove to be helpful in reducing software maintainer's tasks. As the approach is based on analyzing the variables at statement level hence it can produce more precise results. We are currently implementing the tool. So far, variable trace gatherer and database has been completed. Now we are looking to implement the analyzer and graphical interface.

5. REFERENCES

[1] A. Orso, T. Apiwattanapong, J. Law, G. Rothermel, and M. J. Harrold. 2004. An Empirical Comparison of Dynamic Impact Analysis Algorithms. In Proceedings of the 26th International Conference on Software Engineering (Scotland, UK, May 23-38, 2004). (ICSE 2004), 491-500.

[2] A. Orso, T. Apiwattanapong, and M. J. Harrold. 2003. Leveraging Field Data for Impact Analysis and Regression Testing. In Proceedings of ACM Symposium on Foundations of Software Engineering (Helsinki, Finland, September 01-05, 2003). (SIGSOFT 2003), 128-137. DOI = http://doi.acm.org/10.1145/566172.566182

[3] B. Korel and J. Laski. Dynamic Slicing in Computer Programs. 1990. Journal of Systems Software, 13(3) (November 1990), 187–195.

[4] Glenford J. Myres. 1979. Art of Software Testing. John Wiley & Sons, New York, ISBN 0471043281

[5] J. Law and G. Rothermel. 2003. Incremental Dynamic Impact Analysis for Evolving Software Systems. In Proceedings of 14th IEEE International Symposium on Software Reliability Engineering. (Denver, Colorado, Nov 17-20, 2003). (ISSRE 2003).

[6] J. Larus. 1999. Whole Program Paths. In ACM Proceedings of 1999 Conference on Programming Language Design and Implementation (Atlanta, GA, May 01-04, 1999). (SIGPLAN PLDI 1999), 1–11. DOI = http://doi.acm.org/10.1145/301631.301678

[7] J. Law and G. Rothermel. 2003. Whole Program Path-Based Dynamic Impact Analysis. In Proceedings of the International Conference on Software Engineering (Hilton Portland, Oregon, USA, May 3-10, 2003). (ICSE 2003), 308-318

[8] J. P. Loyall, S. A. Mathisen, and C. P. Satterthwaite. 1997. Impact Analysis and Change Management for Avionics Software. In Proceedings of IEEE National Aerospace and Electronics Conference, Part 2 (Dayton, OH, July 1997). 740–747

[9] Mark Sherriff and Laurie Williams 2008. Empirical Software Change Impact Analysis using Singular Value Decomposition. In proceedings of 1st IEEE International Conference on Software Testing, Verification, and Validation (Lillehammer, Norway, April 09-11, 2008). (ICST 2008), 268-277.

[10] M. Kamkar. 1995. An Overview and Comparative Classification of Program Slicing Techniques. Journal of Systems Software, 31(3) (December 1995), 197–214.

[11] M. Lee, A. J. Offutt, and R.T. Alexander 2000. Algorithmic Analysis of the Impacts of Changes to Object-oriented Software. In Proceedings of the Technology of Object-Oriented Languages and Systems (July 30-August 03, 2000). (TOOLS 34'00), 61

[12] R. J. Turver and M. Munro. 1994. Early Impact Analysis Technique for Software Maintenance. Journal of Software Maintenance: Research and Practice, vol. 6 (1), (Jan 1994), 35–52.

[13] R. S. Arnold and S. A. Bohner 1993. Impact analysis – Towards a Framework for Comparison. In Proceedings of IEEE International Conference on Software Maintenance (Montreal, Que, Can, Sept. 1993). (ICSM 1993), 292– 301.

[14] S. Bohner and R. Arnold. 1996. Software Change Impact Analysis. IEEE Computer Society Press (Los Alamitos, CA, USA 1996), 376

[15] S. L. Pfleeger. 1998. Software Engineering: Theory and Practice. Prentice Hall, Englewood Cliffs, NJ

[16] Taweesup Apiwattanapong, A. Orso and Mary Jean Harrold. 2005. Efficient and Precise Dynamic Impact Analysis Using Execute-After Sequences. In ACM International Conference on Software Engineering (St. Louis, Missouri, USA, May 15-21, 2005). (ICSE 2005). DOI = http://doi.acm.org/10.1145/1062455.1062534