CIASYS - Change Impact Analysis at System Level

Gabriella Tóth, Csaba Nagy, Judit Jász and Árpád Beszédes
University of Szeged
Department of Software Engineering
Árpád tér 2. H-6720 Szeged, Hungary
{gtoth,ncsaba,jasy,beszedes}@inf.u-szeged.hu

Lajos Jenő Fülöp
FrontEndART Ltd.
Zászló utca 3. I/5. H-6722 Szeged, Hungary
flajos@frontendart.com

***Abstract*—The research field of change impact analysis plays an important role in software engineering theory and practice nowadays. Not only because it has many scientific challenges, but it has many industrial applications too (*e. g.,* cost estimation, test optimization), and the current techniques are still not ready to fulfill the requirements of industry. Typically, the current solutions lack a whole-system view and give either precise results with high computation costs or less precise results with fast algorithms. For these reasons, they are not applicable to large industrial systems where both scalability and precision are very important. In this paper, we present a project whose main goal is to develop an innovative change impact analysis software-suit based on recent scientific results and modern technologies. The suite will use hybrid analysis techniques to benefit from all the advantages of static and dynamic analyses. In addition, it will be able to determine the dependencies at system level of software systems with heterogeneous architecture. The software is being developed by FrontEndART Ltd. while the theoretical and technological background is provided by the Department of Software Engineering at the University of Szeged. The project is funded by the Economic Development Operational Programme, New Hungary Development Plan.**

***Keywords*-change impact analysis, static analysis, dynamic analysis, hybrid analysis, system-level dependencies**

I. Introduction

The main goal of *impact analysis* is to reveal which other systems or parts of a software system are influenced by some selected parts of a given system. This is usually closely related to *software change impact analysis* [1] which is defined as identifying the potential consequences of a change, or estimating what needs to be modified to accomplish a change. These consequences of a change are usually associated with the risks of a modification in the software. Impact analysis is important for many different purposes, including estimating the cost of development, the cost of a bugfix, the stability of the system, or locating the potential bugs in the system.

There is a number of reasons why an existing system has to be changed. For example, fixing defects, introducing new features, preparing the system for a new environment or the improvement of internal quality. All these changes have potential risk factors because of unexpected side-effects, errors or mistakes. To avoid these risks, the developers estimate the potential consequences of the change (e.g. which parts of the source code must be rewritten or retested) and implement the changes only after their estimation. Due to the risks mentioned above, the cost of a software change is usually very high, which indicates the importance of change impact analysis. This is especially the case for those systems that are already in operation. For live systems, the cost of a bugfix (including development cost, business loss, etc.) might be as much as one hundred times more than a fix of a bug which is discovered in the development phase of the product.

The most important application fields of impact analysis include estimating the cost of a change and decreasing these costs, analyzing the impact of a change to discover the potential bugs or errors, and optimizing software testing especially in terms of resources and efforts. Taking the latter as an example, after modifying the system it needs to be retested (regression and confirmation testing). However, it is important to note that testing a whole system is usually expensive and sometimes not even affordable after each change (sometimes a full test is even more expensive than the development itself). In such cases, it is useful to identify which test cases are required to be rerun in order to test exactly those parts of the system that are influenced by the change (*selective testing*). Identifying these parts of the system requires impact analysis.

Experience shows that developers usually perform impact analysis in an *ad hoc* manner, “manually”. Thus, developers rely only on their skills and experience resulting in the fact that when there is a change request, it is evaluated by an expert who estimates the potential consequences and makes decisions accordingly. The experience and the skills of an expert are very important, but manual work always includes potential mistakes. Moreover, it is very expensive too. Consequently, a tool that is able to (partly) automate the process of impact analysis is very useful in different phases of software development.

In this paper, we give a brief overview of a project co-financed by the European Union through the New Hungary Development Plan, which implements new impact analysis techniques and validates them, thus enhancing the state-of-the-art in this field. The project is carried out in an industrial context to develop an innovative impact analysis software-suite.

II. State of The Art

Automated impact analysis methods based on source code employ the analysis of the relations between different software components (modules, procedures, statements, variables, etc.). With precise analysis, the relations between the lower levels of a system (e.g the relations between statements and data) can be computed. This kind of program analysis is often referred to as *program slicing*. However, this affects the cost of computations and the complexity of impact analysis. On the other hand, the analysis of higher level dependencies (e.g. the dependencies between procedures) are less precise, but faster.

There are *static* and *dynamic approaches* too. Static methods are applicable to the systems without executing them. Thus, one can observe the effect on all the potential executions of a system with one analysis, while with dynamic analysis one can observe only specific executions of the system. Hence, dynamic analysis is usually more precise than static analysis, even if multiple executions of the system are taken into account. Experience shows that the current static techniques do not scale well for the impact analysis of large industrial systems. One explanation is the imprecision caused by the modern dynamic languages (polymorphism, reflection). In such cases, one potential solution is to use *hybrid techniques*, where the imprecise results of static analysis are improved with the precise results of dynamic analysis.

Hybrid and higher level methods are potential solutions for the weaknesses of static analysis and dynamic analysis. However, these methods are not yet elaborated in sufficient detail.

Another challenge for these methods is that if a technique needs to be applied for large systems safely (so, it does not omit any important dependency), it must be able to analyze the system as a whole. However, large modern systems are heterogeneous and their architecture is built up of many subsystems using different technologies. For instance, it is common to use relational databases and access the database by using SQL instructions embedded in the source code.

On the global market, only a few companies provide complete solutions for impact analysis. As an innovative and motivated research area, some of the companies are closely related to universities. For instance, Axivion as a spin-off of the University of Stuttgart [2] provides software development services within the Bauhaus project, including code quality management. Absint [3] provides static program analysis solutions for embedded systems. A prominent company in the fault detection area is Coverity [4] with their best known product Coverity Prevent. Other companies are Fortify Software [5], Klocwork [6] and Parasoft [7] providing solutions for software security assurance too. Grammatech's CodeSurfer [8] is one of the leading tools in the slicing area. Finally, Scientific Toolworks's [9] Understand tool is also notable in the field of impact analysis.

Currently, there is no available tool which is able to provide services and features like those planned to be the outcome of this project. University prototypes are available however, for example Chianti [10] which is an impact analysis tool running in Eclipse environment for Java programs. Similarly, JRipples [11] also runs in Eclipse environment and assists developers in identifying components to be inspected during change analysis. These tools are based on static analysis only, and their applicability is limited for large systems.

III. The Project

A. General data

The project is co-financed by the European Union and the partner support of the Hungarian Regional Development Fund under the "Support of enterprise innovation" tender of the *Economic Development Operational Programme, New Hungary Development Plan*.

The name of the project is "*Development of an impact analysis software package which improves the cost estimation of changes in large heterogeneous architecture software and improves efficiency of testing*," the identifier is *GOP-1.3.1-07/1-2008-0013*.

The total budget of the project is 233 500 EUR, and the amount of funding is 116 750 EUR (50%).

The duration of the project is two years from 1^{st} October, 2008 by 30^{th} June, 2010.

As an outcome of this project, an impact analysis software package will be developed. This software package consists of five main components: an evaluation environment; a static, a dynamic, a hybrid, and a system-level impact analysis tool.

B. Participants of the project

The participants of the project are FrontEndART Ltd. – a spin-off company of University of Szeged –, the Department of Software Engineering at University of Szeged, and subcontractors for different subtasks.

FrontEndART Ltd. plays an important role in source code based software quality analysis in Hungary and globally as well. The best known product of the company is a software quality assurance framework (*SourceInventory*) based on the *Columbus technology*. This technology approaches source code quality from many different aspects: metrics, violations of coding conventions, and code duplications, for instance. All the information is extracted from the source code and uploaded into a database which makes the extracted data available for further automatic processing.

FrontEndART Ltd. co-ordinates the project and the development of its main outcome, the product itself. The Department of Software Engineering, University of Szeged provides the theoretical and technological background; it conducts research, investigates new techniques, and improves on previously published approaches.

C. Main tasks of the project

Figure 1 illustrates the main tasks of the project. First we set up an evaluation environment to test and evaluate our methods. Then, we implement static and dynamic techniques that will be combined in a hybrid analysis system. This system will be able to analyze the dependencies of large systems at system level.

1) Evaluation environment: The first step of the project is to set up an evaluation environment. It is important to evaluate the implemented methods and to measure their effectiveness and efficiency. This environment is a set of open source software focusing on products with available source code of different versions, and provided with suitable test environments.

2) Static impact analysis software: After setting up an evaluation environment, we start developing a static impact analysis software. This software implements new techniques besides evaluating and improving the state-of-the-art methods in the field of static analysis. For instance, the university partner improves its method with heuristics to find SEA (*Static Execute After*) relations [12], [13], [14] effectively and precisely. According to this relation, one procedure of a system depends on another procedure, if it (or its part) is executed after the other procedure. This relation between the different program elements (e.g. procedures) can be determined with static analysis without executing the analyzed software. The algorithm which calculates these dependencies scales well for large systems, and it is adaptable to a given environment by using different heuristics.

3) Dynamic impact analysis software: The next step is the implementation of a dynamic analysis tool. We improve the Java dynamic slicing method and the DFC (*Dynamic Function Coupling*) method that were previously published by the university partner [15], [16]. DFC is based on the analysis of the function call chains captured from the execution traces of the program where the pairs of functions which are regularly "close" to each other are considered as having an effect on each other. This method is also implemented for the Java and C++ languages.

4) Hybrid impact analysis software: The next step is to develop a hybrid impact analysis tool that combines the advantages and eliminates the disadvantages of the static and dynamic impact analysis tools. First, the undiscovered impacts retrieved by the static methods can be completed with the support of dynamic methods (e.g. the dependencies appearing in program run caused by reflection). On the other hand, the false or less significant impacts discovered by the static methods can be filtered out by taking into account the results of dynamic methods. In addition to the implementation of the tool, we give a statistical model to determine the number of required program executions in order to calculate as many potential dependencies in a program as possible with reasonable accuracy.

5) System-level impact analysis software: The previous methods can only be applied to given modules of a system. Hence, these techniques compute dependencies only between the different elements of the analyzed module, and they are not able to compute the dependencies between the different modules of the full system. We implement methods to compute dependencies in large systems by using databases where dependencies arise through database access, such as the dependencies between columns, tables, procedures, stored procedures, etc. Besides, the software will be able to recover the dependencies between the components of distributed systems and service-oriented architectures.

6) Evaluation: There are a number of different ways to evaluate impact analysis techniques. We evaluate them by using our evaluation environment. We compare the different techniques in terms of applicability, scalability, and efficiency. In addition, we set up typical use cases (scenarios) for cost estimation and testing, then we examine which method should be used in which specific cases.

D. Expected outcomes

The motivation of this project is that despite the increasing demand for impact analysis tools in the software industry the existing techniques are not yet sufficiently mature. Many commercial solutions or prototypes developed for research purposes cannot be effectively used in real environments or for large-scale systems due to the known limitations of static and dynamic analysis techniques.

The main design goal is that the tools developed – compared to existing solutions – be efficiently applicable to large scale industrial systems, namely to be scalable and precise enough at the same time.

We also expect the new software to compute dependencies between the components of distributed and service-oriented systems.

For database-intensive systems, it will be able to determine the dependencies between database components (e.g. columns, tables, stored procedures), and between these elements and other system components (e.g. classes, methods or functions).

IV. Status of the Project and Future Work

After setting up the evaluation environment, we developed the static impact analysis software and enhanced the previously published methods to compute SEA relations [12], [13] with different heuristics [14]. We implemented the methods for Java and C++ programming languages. The enhancement of the method using the heuristics enables more precise results in the sense that it removes certain dependencies, which can be proven to be false. This decision is based on the investigation of what kind of data is used

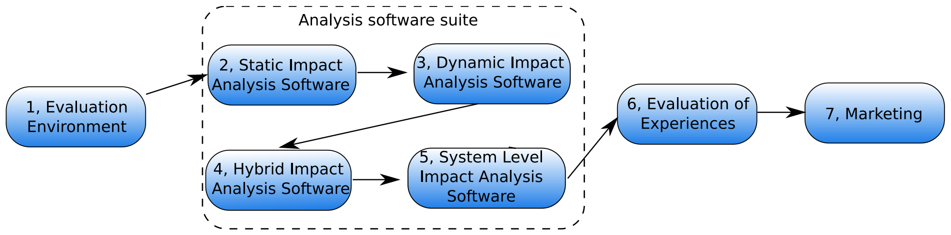
Analysis software suite
1, Evaluation Environment
2, Static Impact Analysis Software
3, Dynamic Impact Analysis Software
4, Hybrid Impact Analysis Software
5, System Level Impact Analysis Software
6, Evaluation of Experiences
7, Marketing

Figure 1. Main tasks of the project.

by the individual procedures: if it only reads data then it cannot induce data dependency, and similarly, if it does not read any data it cannot depend on others.

The evaluation of the tools developed will be done on different levels. First, the basic dependency computation components will be thoroughly tested using the already available evaluation benchmark. Second, the overall working of the system including high level use cases will be validated using the specially developed user interfaces. These interfaces include the integration into the SourceInventory framework of the company and the integration into different development environments as well.

Currently, the development of the dynamic impact analysis software is in progress in parallel to the subtasks of the system-level impact analysis. The university partner has already published results in this area [15], [16], but the improvement of the current methods is still in progress.

As future work, we need to develop the software for hybrid and for system-level impact analysis in order to be able to evaluate and compare the results of different techniques.

Acknowledgements

This research was supported by the New Hungary Development Plan, Economic Development Operational Programme, grant id: GOP-1.3.1-07/1-2008-0013.

References

[1] R. S. Arnold, *Software Change Impact Analysis*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1996.

[2] "Axivion GmbH," http://www.axivion.com/.

[3] "AbsInt Angewandte Informatik GmbH," http://www.absint.com/.

[4] "Coverity, Inc." http://www.coverity.com/.

[5] "Fortify Software Inc." http://www.fortify.com/.

[6] "Klocwork Inc." http://www.klocwork.com/.

[7] "Parasoft Corporation," http://www.parasoft.com/.

[8] "GrammaTech's CodeSurfer," http://www.grammatech.com/products/codesurfer.

[9] "Scientific Toolworks, Inc." http://www.scitools.com/.

[10] X. Ren, F. Shah, F. Tip, B. G. Ryder, and O. Chesley, "Chianti: a tool for change impact analysis of java programs," in *OOPSLA '04: Proceedings of the 19th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*. New York, NY, USA: ACM, 2004, pp. 432–448.

[11] "JRipples tool for Incremental Change," http://jripples.sourceforge.net/.

[12] Á. Beszédes, T. Gergely, J. Jász, G. Tóth, T. Gyimóthy, and V. Rajlich, "Computation of static execute after relation with applications to software maintenance," in *Proceedings of the 2007 IEEE International Conference on Software Maintenance (ICSM'07)*. IEEE Computer Society, Oct. 2007, pp. 295–304.

[13] J. Jász, Á. Beszédes, T. Gyimóthy, and V. Rajlich, "Static execute after/before as a replacement of traditional software dependencies," in *Proceedings of the 2008 IEEE International Conference on Software Maintenance (ICSM'08)*. IEEE Computer Society, Oct. 2008, pp. 137–146.

[14] J. Jász, "Static execute after algorithms as alternatives for impact analysis," *Peryodica Politechnica*, Budapest, 2009, Accepted paper.

[15] A. Szegedi and T. Gyimothy, "Dynamic slicing of java bytecode programs," in *SCAM '05: Proceedings of the Fifth IEEE International Workshop on Source Code Analysis and Manipulation*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 35–44.

[16] A. Beszedes, T. Gergely, S. Farago, T. Gyimothy, and F. Fischer, "The dynamic function coupling metric and its use in software evolution," in *CSMR '07: Proceedings of the 11th European Conference on Software Maintenance and Reengineering*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 103–112.