Automation Of Impact Analysis

T. Jalaja[1], T.Adilakshmi[2]
[1,2]*Department of Computer Science and Engineering,*
Vasavi College of Engineering, Ibrahimbagh, Hyderabad, India

Abstract- In our present business scenario, more than 70% of the projects being executed come under maintenance. Proper and complete Impact analysis on the affected system due to the incremental enhancements plays a significant role in identifying the consequences of the modification. Incomplete impact analysis affects critical business processes and poses a risk to business continuity. Today, there is a strong dependency on application SME (Subject Matter Expert) to do a holistic impact analysis, and there could be a significant time lag in reaching out to SMEs working in a different location or time zone.

The Objective of this research proposal is to design a workflow (driven by BPMN) for automating the impact analysis on the affected system. This tool takes the business requirement in a plain text and provides the impacted list of system components and their complexity in terms of Lines of Code (LOC), Function Points (FP).

This tool can further be extended to identify the regression test suite based on the impacted objects to enable "Shift left testing" and also can generate a traceability matrix, mapping requirement with impacted components and regression test cases.

Keywords – Impact Analysis, NLP API, Lines of Code, Function Points

I. INTRODUCTION

Software products are naturally adapted and changed with the changing system requirements to meet user's needs. Software change is a fundamental ingredient of software maintenance. Software maintenance has been recognized as the most difficult, costly and labour-intensive activity in the software development life cycle [1, 2]. According to Lehman and Belady change is continual [3].Changes might be required due to new requirements, fixing faults, change requests and so on. When changes are made to software, they might be some unexpected effects and may cause inconsistencies with other parts of the original software. Software change impact analysis involves a collection of techniques for determining the effects of the proposed changes on other parts of the software [4]. It plays an important role in software development, maintenance, and regression testing [5].

The rest of the paper is organized as follows. Literature survey on Software Change Impact Analysis is given in section 1.1. The Tool Support available in 1.2. Applications of Change Impact Analysis Techniques are presented 1.3. The Research methodology is presented in section II. Concluding remarks are given in section III.

1.1. Literature Survey -

In the last 35 years, software engineering researchers have proposed several definitions of CIA and developed many techniques for Change Impact Analysis. In 1996, Bohner et al. defined CIA as 'the process of identifying the potential consequences of a change, or estimate what needs to be modified to accomplish a change' in the book Software Change Impact Analysis [4].

There have been wide ranges of research work on CIA techniques and its applications. Some CIA methods are based on the traceability examination (traceability-based CIA) whereas others concentrate on dependence relationships (dependence-based CIA) to determine the change effects. Impact analysis techniques try to estimate the change impact from the analysis of software artifacts. Among other approaches, static and dynamic analysis are the most popular techniques in the literature [6],[7].

From an impact analysis viewpoint, software entities are pieces of design or code that may undergo changes. In object-oriented software, methods/functions and fields are usually the entities at the granularity that is important to analysis. Hence, from an initial change set of entities, the challenge is finding out additional entities affected by the change in this initial set. The extended set with initial and affected entities is called the impact set. To find out these additional entities, relations between entities are used. In the static approach, structural dependencies are identified between software entities from a static representation of the software, usually extracted from source code. In the dynamic approach, dynamic analysis is done by instrumenting source code to record program event traces. Generally, events are method/function calls and returns and variable reads or writes. Temporal relations between events in traces are used to find out impacted entities based on system behavior.

1.1.1 Software Change Impact Analysis

According to [8] Hattori presented a probabilistic software change impact analysis technique that combines IA with MSR to sort the results provided by IA based on historical change information. The technique is composed of: (1)

static change impact analysis technique –identify possible impacted entities given a set of changes; (2) an extractor, responsible for extracting all committed changes from the repository; and (3) a theoretical model, based on Bayes theorem, to combine IA results with information extracted from the repository.

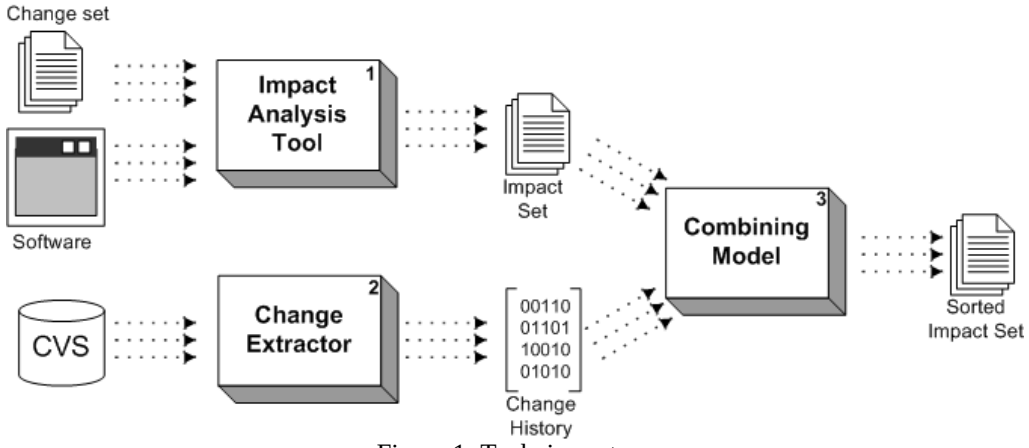

Figure 1: Technique steps

Figure 1 shows the overview of the probabilistic IA technique. The impact analysis tool, called Impala, analyzes specifically object-oriented systems implemented in Java. Impala is responsible for identifying possible impacted entities given a proposed change set to the software. Note that it analyzes the software before the execution of the changes. It takes as input the proposed change set and the software's current source code to give as output an impact set. The change set is composed of a certain number of entities, i.e. class, method or field, to be changed in a modification task. In order to search for possibly impacted entities, Impala first produces from the source code a graph that represents the system. In this graph, the nodes are entities and the arcs are relationships between two entities. These relationships can be of the following types: instanceOf, contains, extends, implements, isAccessedBy, isInvokedBy, and their inverse relations.

After producing the graph, Impala searches for dependencies of each change from the change set individually and according to the change type. The change types can be: add, remove, change or change visibility (public, protected, package or private) of class, method or field, and add inheritance, remove inheritance of class. So, for each change in the change set, Impala exhaustively searches for entities that depend on the entity that will change, according to the change type. For example, if a method will be removed, Impala searches for its callers (inverse relation of isInvokedBy), for the callers of its callers and so on. The result is a list containing the change set and all entities possibly impacted from each change in the change set. Note that this list contains many false-positives, e.g. entities that will not be impacted by a change.

The Change extractor, called Impala Plug-in, is an Eclipse plug-in that extracts from CVS information concerning all and every change to the software in the past.

Impala Plug-in pre-processes CVS data automatically. It accesses the software's CVS and, for each class, extracts all structural changes, organizes these changes into commit transactions, and outputs a matrix containing information of what structural changes were committed together in the past.

Structural change is any syntactic change to the source code, e.g. change to a method's signature, addition or deletion of a class, deletion of an inheritance relationship. Changes in comments, in license term, or formatting changes are not considered structural changes, because they do not modify the functioning of the system. To extract the structural changes, Impala converts every revision of a class into an abstract syntax tree (AST), compares every two subsequent revisions and stores every single structural change that occurred from one revision to another. After extracting all structural changes, Impala Plug-in organizes them into commit transactions. The idea of our theoretical model is to calculate, for each entity in the impact set, the probability of its impact based on the frequency that this entity was committed together with the entities from the change set.

A change made to a software system may result in an undesirable side effect and/or ripple effect [4].The goal of CIA is to identify the ripple effects and prevent side effects of a proposed change. A side effect occurs when changes to the software cause it to reach an erroneous state. The concept of ripple effect is often used to measure the likelihood that a change to a particular module may cause problems in the rest of the program. Figure 2 shows the whole CIA process [9, 10]. CIA starts by analyzing the change request and the source code to identify the change set in which the elements could be affected by the change request. Then through the change impact analysis technique, other elements that are probably affected by the elements in the change set are estimated. The resulting set is called

estimated impact set (EIS). Once the change is implemented to accomplish the change request, the elements in the actual impact set (AIS) are actually modified.

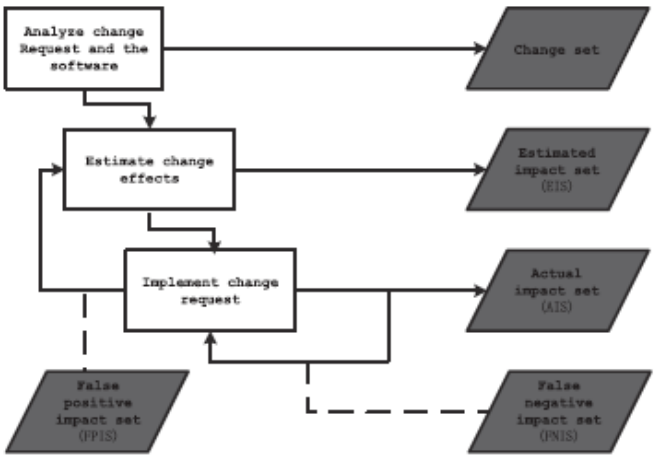

Figure 2: Change Impact Analysis Process

The goal of the CIA process is to estimate an EIS that is as close to the AIS as possible. Several metrics can be defined to evaluate the accuracy of the impact analysis process [11]. Among the metrics for accuracy of CIA techniques, precision and recall are two widely used metrics [11].
Since CIA plays a crucial role in software maintenance and evolution, its automation is necessary to encourage wider adoption. The decision to select a particular CIA technique may be affected by the availability of tool support. Existing CIA techniques have varied from supporting traditional procedure-oriented programs to object-oriented programs. In recent years, a lot of work has been studied to support impact analysis for object-oriented programs. These Impact Analysis techniques take concrete features of the object-oriented programs into account and generate the potentially impacted classes, class methods and/or class fields.

1.2 Tool support-
In spite of so many surveyed CIA techniques, there are only 10 available tools to support these techniques. In [12] John Wiley had listed these tools in Table 1, together with their required input data, output and the programming languages supported.

Tool support			
Name	Input	Output	Language
Columbus	Object-oriented system	Structural coupling measures	C++
ROSE	Software historical repositories; current program; changes	Impacted parts	Java
EAT	Execution information; proposed changed methods	Impacted methods	Java
Sieve	Programme binaries of original and modified program	Impacted methods and code regions in modified program	*C*
Jimpa	A change request description; historical source files repositories	Impacted files	Any*
JDIA	Changes; the program; some executions	Impacted methods and fields	Java
Impala	The system; changes	Impacted elements	Java
IRC²M	A project	Conceptual coupling measures	Any*
JRipples	The system; changed classes	Impacted classes	Java
LDA	A software project	Relational topic based coupling	Any*

Table 1: Tool support of the change impact analysis techniques

Some tools such as Impala and JRipples assist in static analysis of the current system. Impala supports CIA before the change implementation [8]. Its input includes classes, methods and fields. Impact analysis is started by searching in the dependency graph from the elements in the change set according to the change type and the dependency type. Those elements, which are reachable from the elements in the change set, are collected as the output of this tool. JRipples supports program comprehension during incremental changes [12]. It analyzes the program and automatically marks the impacted classes. Later when a modification is designated, other impacted classes will be automatically shown.

There are also some tools, such as EAT and JDIA, which use the execution information to support dynamic CIA. EAT uses execute-after sequences [6]. Its input is the changed methods and execution information, whereas its output is the dynamic impact set at the method level. JDIA works on object-oriented programs. It takes the changed entities, the program, and some executions as the input, and outputs the impacted methods and fields, which can be saved as text file or XML file.

In addition, some tools such as ROSE [13] and Jimpa [14], have been developed to analyze the software repositories for impact analysis. ROSE is a plug-in for the eclipse environment. It analyzes the version history and the given changes, and outputs the likelihood that further changes should be applied to the given location. Jimpa is also a plug-in, which starts from a change request description and the set of historical source file revisions. Then based on an information retrieval approach, some impacted files are identified by referencing similar past change requests.

CIA techniques can be classified from four perspectives: traditional dependency analysis, software repositories mining, coupling measurement and execution information collection.

1.3 Applications of Change Impact Analysis Techniques –

Change impact analysis plays an important role in software development, maintenance and regression testing [4-5, 15]. During the survey, many applications of CIA in software maintenance were found, that is, software comprehension, change propagation, selection of regression test cases, debugging and measuring various software attributes such as changeability and maintenance cost.

Thus, CIA can indirectly support maintainers to make decision among various change solutions, prepare change schedule, estimate resources and costs, and trace the change effect. Table 2 lists the main application areas of CIA and associated tools support (if available).

Application area	Tools support
Software comprehension	*JRipples* *ROSE* N/A
Change propagation	*JTracker* *JRipples* N/A
Regression testing	*Chianti* *Celadon* N/A
Debugging	*Crisp* *AutoFlow* *Chianti*
Software measurement	*ROSE* N/A

Table 2: Change Impact Analysis Applications

1.3.1 Software comprehension –

Before a change can be implemented, a detailed knowledge of the system must be acquired to determine where and how to make the required change. CIA can assist in understanding the original programs. If the potential effects of changes can be calculated before the changes are implemented, the programmer can accurately and efficiently perform the required changes.

1.3.2 Change propagation –
Change propagation is a major application of CIA. As changes are made, maintainers must ensure that other entities in the software system are consistent with these new changes. Change propagation is defined as the changes required of other entities of the software system to ensure the consistency of assumptions within the system after a particular entity is changed [16]. It is often performed during incremental changes [17]. Impact analysis is first used to predict the change effects, which can then be checked to see whether they need modification.

1.3.3Regression testing –
Change impact analysis can also be used for regression testing by selecting the test cases that need to be rerun and adding new test cases to cover the changes and affected parts not tested by the original test suite. This will help build confidence in the integrity of the system after modification.

1.3.4 Debugging –
Programmers often spend a significant amount of time debugging programs to locate the faults and make correction. Typically, CIA techniques can be employed when regression tests fail unexpectedly by isolating a subset of responsible changes for that failing test. Then, failure-inducing changes can be found out more effectively. Chesley et al. proposed a tool Crisp for debugging Java programs.

II. RESEARCH METHODOLOGY

2.1 Research Methodology –
The Objective of this research proposal is to design a workflow (driven by BPMN) for automating the impact analysis on the affected system. This tool takes the business requirement in a plain text and provides the impacted list of system components and their complexity in terms of Lines of Code (LOC), Function Points (FP).

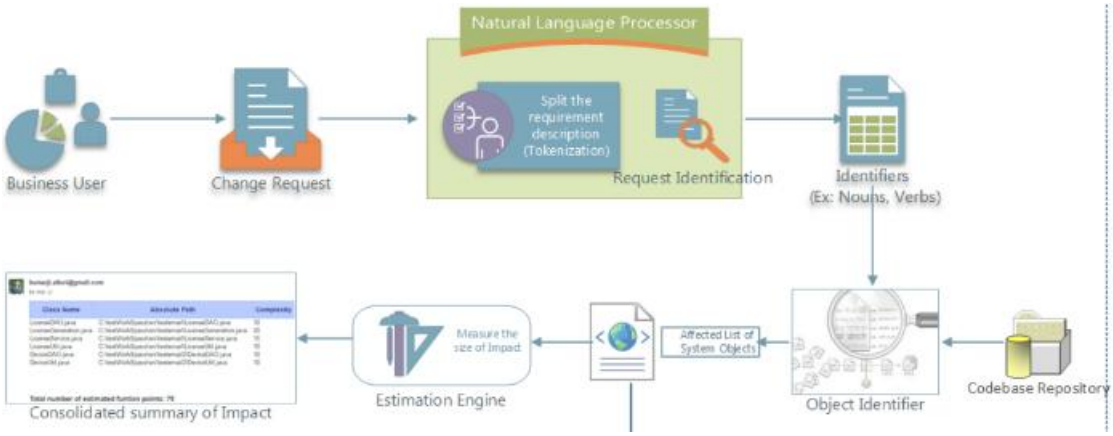

Figure 3: Impact Analyzer Workflow

Natural Language Processor (Open NLP API) can be used to parse the requirement text and identify the key identifiers (from the requirement). Object Identifier referred in the above workflow (Figure 3) takes the identified keywords, connects to the application source code repository, performs the wildcard search and identifies the list of the affected system objects. The affected list of system objects and their methods will be considered as impacted ones and the complexity of the change request will be calculated based on the metrics like Lines of Code (LOC), Function Points (FP) on the impacted objects.
This tool can further be extended to integrate with regression testing tool (by passing the list of impacted objects) and can generate the traceability matrix mapping requirement with impacted components and regression test cases.

III. CONCLUSION

As per Bixin Li [18] though some tools have been developed to support CIA, most of these tools are just prototypes. No CIA tools have been commercialized. Since CIA is increasingly pivotal, good tool support is in great need. A few tools such as JRipples [12] have approached maturity and stability. These tools should be fully evaluated, promoted and refined based on industrial use. Additional tools should be developed to support all aspects of CIA.

The Proposed tool provides an end to end solution in automating the impact analysis process by identifying the list of impacted application components and the corresponding regression test suite. It helps the business analysts and client coordinators to get a fair idea on the impact of the change instantly. The Automated tool helps to remove the tight dependency with SMEs. It also provides the ability to justify the impact to business effectively. Zero effort is required from application teams for adoption of the tool.

IV. REFERENCES

[1] Li W, Henry S. Maintenance support for object-oriented programs. Journal of Software Maintenance: Research and Practice 1995; 7(2):131–147.

[2] Schneidewind NF. The state of software maintenance. IEEE Transactions on Software Engineering 1987;13(3):303–310.

[3] Lehman MM, Belady LA. Program Evolution: Processes of Software Change. Academic Press: London, 1985.

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

[5] Rovegard P, Angelis L, Wohlin C. An empirical study on views of importance of change impact analysis issues.IEEE Transactions on Software Engineering 2008; 34(4):516–530.

[6] T. Apiwattanapong, A. Orso, and M. J. Harrold, "Efficient and precise dynamic impact analysis using execute-after sequences," in ICSE '05: Proceedings of the 27th international conference on Software engineering. New York, NY, USA: ACM, 2005, pp. 432–441.

[7] S. B. Robert Arnold, Software Change Impact Analysis. Los Alamitos, CA, USA: IEEE Computer Society Press, 1996.

[8] Lile Hattori , Gilson dos Santos Jr., Fernando Cardoso, Marcus Sampaio -"Mining Software Repositories for Software Change Impact Analysis: A Case Study" Proceedings of the Brazilian Symposium on Databases, Campinas, Brazil, 2008; 210–223.

[9] De-Lucia A, Fasano F, Oliveto R. Traceability management for impact analysis. Proceedings of the International Conference on Software Maintainence, Beijing, China, 2008; 21–30.

[10] Bohner SA. Software change impacts—an evolving perspective. Proceedings of the International Conference on Software Maintenance, Montréal, Canada, 2002; 263–272.

[11] Hattori L, Guerrero D, Figueiredo J, Brunet J, Damsio J. On the precision and accuracy of impact analysis techniques.Proceedings of the Seventh IEEE/ACIS International Conference on Computer and Information Science, Portland, Oregon, 2008; 513–518.

[12] Buckner J, Buchta J, Petrenko M, Rajlich V. JRipples: a tool for program comprehension during incremental change. Proceedings of the International Workshop on Program Comprehension, St. Louis, MO, USA, 2005;149–152

[13] Zimmermann T, Zeller A, Weissgerber P, Diehl S. Mining version histories to guide software changes. IEEE Transactions on Software Engineering 2005; 31(6):429–445.

[14] Canfora P, Cerulo L. Jimpa: an eclipse plug-in for impact analysis. Proceedings of the Conference on Software Maintenance and Reengineering, Bari, Italy, 2006; 341–342.

[15] Bohner SA. Impact analysis in the software change process: a year 2000 perspective. Proceedings of the International Conference on Software Maintenance, Monterey, CA, USA, 1996; 42–51.

[16] Hassan AE, Gall RC. Replaying development history to assess the effectiveness of change propagation tools.Empirical Software Engineering 2006; 11(3):335–367.

[17] Rajlich V, Gosavi P. Incremental change in object-oriented programming. Empirical Software Engineering 2004;21(4):62–69.

[18] Bixin Li, Xiaobing Sun, Hareton Leung and Sai Zhang:A survey of code-based change impact analysis techniques, 29 March 2012.