A Meta Level Dynamic Approach to Visualize Impact Analysis for Regression Testing

Omid Pourgalehdari, Kamal Z. Zamli, Nor Ashidi Mat Isa
School of Electrical and Electronic Engineering
Universiti Sains Malaysia, Engineering Campus, Penang
Malaysia
Email: {eekamal, ashidi}@eng.usm.my

Abstract

Fixing software bugs, extending base applications with new functionalities, as well as adapting to changing environments are among the reasons for software evolution. To facilitate such a process and to help maintainers make informed decision, there is a need to be able to estimate and determine the impacts of evolution to the overall software system. While there are already quite a number of useful tools, termed impact analysis tools, developed either as research prototypes or commercial products that addresses such issues, much of which is static based and adopts traditional text based impact reporting. To address some of these issues, we propose a new change impact analysis, called JRegres, that can dynamically generate trails for impact analysis as well as support impact visualization. JRegres serves as a research vehicle to investigate the hypothesis that suggests impact visualization is useful for supporting regression testing.

I. INTRODUCTION

Software evolution is an integral part of the software lifecycle. Fixing software bugs, extending base applications with new functionalities, as well as adapting to changing environments are among the reasons for software evolution. To facilitate such a process and to help maintainers make informed decision, there is a need to be able to estimate and determine the impacts of evolution to the overall software system. While there are already quite a number of useful tools, termed impact analysis tools, developed either as research prototypes or commercial products that addresses such issues, much of which is static based and adopts traditional text based impact reporting. To address some of these issues, we propose a new change impact analysis, called JRegres, that can dynamically generate trails for impact analysis as well as support impact visualization. JRegres serves as a research vehicle to investigate the hypothesis that suggests impact visualization is useful for supporting regression testing.

This paper organized as follows. Section 2 presents some related work. Section 3 discuss on different available techniques on each part of tool and end this section will explain how our approach can handle change impact analysis section 4 will discuss on conclusion and section 5 will be list of references.

II. ISSUES ON IMPACT ANALYSIS

Change impact analysis has been practiced in various forms for years, but yet there is no consensus definition. Pfleeger and Bohner [2] define change impact analysis as “the evolution of many risks associated with change, including estimates of the effect on resources, effort, and scheduler”. Turver and Munro [1] define change impact analysis as “the assessment of change to a source code of a module on other module of system. It will determine the scope of a change and provides a measure of its complexity”.

Concerning classification, Lee [3] categorizes change impact analysis based on the dominant techniques used (described below)

- Data flow analysis focuses on the level of impact of data on the program.
- Path impact analysis is a dynamic analysis that obtains data from the
execution and trace of program functions to calculate the impact sets.

- Data dependency analysis [9,10] focuses on the static impact information that captured from source code.
- Control flow analysis [8,9] identifies calling dependencies, logical decisions and other control information.
- Program slicing [7,8] captures snapshot of essential parts of the program by eliminating parts that are do not contribute for the evaluation of the specific variables at a certain location.
- Test coverage analysis [5] analyzes the program coverage in terms of capturing functions which are called during execution (i.e. combining the slice techniques for calculating the impact sets).
- Cross referencing analysis [2] captures parts of a program contain references to a given variable or procedure.
- Reverse engineering [12] that is the process of deriving abstract formal specifications from the source code of the legacy systems where these specifications can be used to forward discovering the technological principles of a device or object or system through analysis of its structure.

Concerning implementation, Appleton [15] identifies nine specific problems/requirements for the current traceability practices. Among others are unnecessary creations of trace artifacts, inevitable failure, too many up front activities, and the need for comprehensive documentation.

In order to address such issues, we believe that visualization techniques can help. With visualization, we can potentially have multiple views of many items of interest from different perspectives. Here, some mapping can be defined. For instance, artifact centered mapping can focus on artifacts. Dependency centered mapping can stress more on the dependencies.

Furthermore, using visualization, it may be straightforward to pin point any irregularities from items of interests. As such, we believe this can certainly helps to reduce the complexity of the current traceability practices.

A number of visualization tools can be adopted for our work. Graphviz is a tool by AT&T [12] for drawing graphs can be adopted should we decide to visualize our impact analysis as connected arcs and nodes. Apart from Graphviz, there are also graph drawing tools available such as Marshall et al.’s graph visualization framework (GFV) [13]. Polaris [14] a tool to visualize database relations may also be another option for our work. ILOG discovery [16] could be another option as it gives the ability to use declarative construction of data linear visualization like bar graphs and histograms. In this regard, we have yet to finalize our implementation.

III. ISSUES ON IMPACT ANALYSIS

To ensure effective implementation, we opt for dynamic analysis techniques. Unlike static analysis techniques which are essentially code walk through module level relationship like class or component diagrams, dynamic analysis can reveal hidden dependencies, for instance, dynamic instantiation of objects at runtime.

According to Ben Breech et al. [11] dynamic impact analysis techniques are divided in two main groups. The first groups are those techniques which obtain data by instrumenting the source code or byte code to calculate impact sets in execution time. The limitation of the approach adopted by these groups is the fact that they need access to the source code. In this respect, impact analysis cannot be performed in the absence of source code.

The second groups are techniques based on the use of a dynamic compiler that obtain the data from crosscutting on run time. In this approach, the source code need not be instrumented. Thus, this approach can be useful for tracing out COTs (i.e. source codes are often unavailable). Here, AspectJ is a useful extension for our work. Based on Java, , it is possible to trace method calls from multiple classes with AspectJ.

As highlighted earlier, JRegress works at the meta level. By working at the metal level, JRegress can intercept all function calls made by the program at runtime and produce the required trails for analysis. Of course, the use of the meta level will only be applicable to only interpreted language such as Java or Smalltalk.
IV. OUR APPROACH

Our approach can be summarized in Figure 1. The main components of the tool are as follows (as enclosed by the bold dotted line):

- Trace monitor performs the monitoring of the method calls at the meta level.
- Dynamic trace is our special database for storing the impact trails.
- Converter analyzes the log file and converts into some form format to be understood by the visualizer.
- Visualizer is the actual component that permits the visualization of the trace. It maps the impact trace into some form of mapping.

Here, the user first interacts with class files. It should be noted that it is possible that the source codes for the class files are absent. As a result of the interaction of the user, traces will be saved in the dynamic trace. To permit visualization, the user can interact with the visualizer, which in turn solicits the converter to perform the mapping.

V. CONCLUSION

Dynamic trace ability and analysis capability within software modules and components can be helpful to help reduce maintainer’s effort. This feature can help to identify the impact codes enabling good traceability coverage.

We are currently implementing the tool in our laboratory. So far, the trace monitor component and the dynamic trace database has been completed. We are now looking to implement the converter and visualizer.

ACKNOWLEDGMENT

This work is partially funded by the eScience Fund – “Development of a Software Fault Injection Tool to ensure Dependability of COTS for Embedded System Applications”.

REFERENCES

Journal of Software Maintenance Research and


