VISUALIZATION OF IMPACT ANALYSIS SOFTWARE AS A TOOL OF REVERSE ENGINEERING TECHNIQUE

Rita Noremi Bt Mohamad
Centre for Advanced Software Engineerering (CASE)
Fakulti Sains Komputer dan Sistem Maklumat
Universiti Teknologi Malaysia, City Campus
54100 Kuala Lumpur
rit_emi78@yahoo.com

ABSTRACT

Most of the system is not 100% bugs free and need to maintain. These make the development team spent their time to read the entire source code that might be have more than 100,000 line of code to debug the error that exist in the system. Reverse Engineering technique will be use as the tools to assist the development team to understand the source code. Through the reverse engineering, the source code will be parsed to capture and identify the impact between artifacts via dependency. To more easy understand the source code, visualization will use as the supporter tools for impaction. In this paper, a researcher will focus on combination of Change Impact Analysis (CIA) and visualization later call Visualization of Impact Analysis (VIA) to assist development team do the system maintenance easier and efficient based on Object-Oriented Programming (OOP) in C++ language as a model to apply in the prototype.

KEYWORDS

Visualization, Change Impact Analysis, Object-Oriented Programming, Reverse Engineering Technique, Ripple-effect

1. Introduction

Change Impact analysis identifies the consequences or ripple-effects of software changes. The recent Year 2000 (Y2K) date phenomenon was a good example of why change impact analysis is needed to identify impacts of software change [1]. In spite of that, visualization also is a good support tool to help visualize the impact of code before software change takes a place. As these software changes are introduced, avoiding defects becomes increasingly labor intensive and error prone. By identifying potential impacts before making a change, we can greatly reduce the risks of embarking on a costly change because the cost of unexpected problems generally increases with the lateness of their discovery. Software maintenance process is one of the most costly activities within information system practice. It has been estimated to account for 50% or more of the total development cost, and this maintenance cost shows no sign of declining [2]. Essential to the software maintenance process is the ability to understand not only the software but also the required visualization as well. This requires the ability to derive software knowledge from the affected source code.

Unlike other type of products, software products are intended to be adaptable. Unfortunately, a seemingly small change can “ripple” throughout the system to have major unintended effects elsewhere. As a result, software developers need mechanisms to understand the existing software and its dependencies. Change Impact analysis (CIA) is regarded a study of component relationships and their ripple effects within software [3]. It makes the potential effects visible before the changes are implemented. The ability to visualize the program dependencies will greatly help a maintainer or management to determine appropriate actions to take with respect to decision making. Turver and Munro [4] define impact analysis as “the assessment of a change, to the source code of a module, on the other modules of the system. It determines the scope of a change and provides a measure of its complexity”. Arnold and Bohner [5] define impact analysis as “identifying potential consequences of a change, or estimating what
needs to be modified to accomplish a change”.
Both definitions emphasize the estimation of the
impacts since what was actually changed is not
fully known until after the software change is
complete.

In a software evolutionary environment, there
may exist an issue of making software changes
in the source code while the rest of the work
products remained not up-to-date. As the source
code remains the most reliable artifact, it is
clearly accepted by many researchers that codes
are the most central reference point to
maintenance tasks. It is a great challenge to
capture all the required software knowledge
available in the codes and presents it in the way
that is easily understood by the maintainers. We
need a special mechanism how to handle
visualization of ripple-effects i.e. by changing a
method; one can visualize the impact of which
method it calls and which method call it.

2. Background
Research related to this work primarily comes
from the source code based maintenance of
object-oriented software. Other related, but less
significant areas such as design recovery,
architectural design and design patterns are not
included in the discussion.

CodeSurfer is the one of Reverse Engineering
tools that provide program analysis,
understanding and inspection system for ANSI
C. CodeSurfer was produce by GrammaTech’s
which based on system dependence graph, a
fundamental intermediate structure for
representing program[6]. This product was focus
on structure programming and dependency based
on methods, variable and statements. Then
CodeSurfer represent all the information in
visualization.

CC-RIDER was produce by Western West is
similar like CodeSurfer but CC-RIDER focus on
C and C++ especially object-oriented
programming. This product is a visualization
tools that present complete information on
functions, variables, enum values, macros and
e tc [7]. But both products more on visualize the
interaction of relationship between components.

This research differs from the above. In that
researcher try to identify the ripple effect of a
software component of interest and turns it into
visualization. It’s mean a user can visualize the
dependencies between software components
which can be defined in terms of variables,
methods or classes and it is called as the software
artifacts. The knowledge of artifact relationships
needs a special study on impact analysis that
relates to ripple-effects before it can be turned
into graphical views for visualization.

3. Ripple-Effects
Fundamental to the software change process is a
set of proposed changes to some software
artifacts and their dependencies that directly
reflect the change requests. To simplify the task,
components to be changed need to be defined at
the current artifact level. For example at the
source code level, the component to be changed
is defined as a class, method, declaration or
object and also its type of change.

The types of change can be generally classified
as update, creation, or deletion. Creation can be
either an addition or insertion of a new
component with its new relationships into the
program. Deletion is considered as discarding a
component with its existing relationships in the
program. While update is applied to an existing
component and its does not change its
relationships with others. Once a change is made
to the software, a maintainer would like to know
its potential impacts on other parts of the
program. This process is called ripple-effects.
For a given proposed change component and its
type of change, a further ripple-effect or
potential change impact can be identified along
with its relevant dependencies [8]. Then the
potential change impact is regarded as a new
initiating proposed change to identify further
change impact. This process continues until no
more further change impact can be identified.

4. Proposes Visualization of Impact Analysis
This research allows a software change to be
made on components such as classes, methods,
and variables. This research also seeks the
support to understand the impact of
dependencies component by visualizing the
impact in Object Oriented Programming. To
simplify the change process, this research
assumes that a change request has already been
translated and expressed in terms of design-level
or code-level items.
The steps of change impact analysis are as follows:

i) Create models of dependencies among software components.

ii) Translate a proposed software change into the impacted elements and relationships.

iii) Trace relationships and reasonably bound the search for the impact of changes, and

iv) Retranslate the impacted elements into visualization.

Before the change impact analysis takes place, firstly we need to extract some dependence graphs from source codes using the regular expression provided by JAVA. Secondly, we need to construct a knowledge-base based on the dependence graphs and store them into a repository.

4.1 Granularity Level

The ripple-effects can be implemented at every level of granularity. This work defines three hierarchical levels of granularity:

a) class level

b) method level

c) variable level

For example at the method level, the ripple-effects process can be used to identify potentially affected methods due to a change in one method. The ripple-effects process should also be performed across different levels of granularity. For example, a change made to a variable may affect the methods containing this variable, and the classes containing this method.

Three types of program slicing (Table 1) are used to locate the transitive closure of certain relationships among the components in order to determine the ripple-effects. Program slices focus attention on small parts of the program by eliminating parts that are not essential for the evaluation of the specific component at a certain location.

<table>
<thead>
<tr>
<th>Type of slicing</th>
<th>Type of slices</th>
</tr>
</thead>
<tbody>
<tr>
<td>Class slicing</td>
<td>This slice is a set of class names</td>
</tr>
<tr>
<td>Message slicing</td>
<td>This slice is a set of messages</td>
</tr>
</tbody>
</table>

Depending on the complexity of the software and the initial change, one can start ripple-effects at the class level to get the overall understanding of the affected parts and proceed to method and then variable or statement level to identify the exact parts affected by the change.

Thus, the ripple-effects for OO programs can be a hierarchical process. Different types of slicing technique are required to perform the ripple-effects analysis at different levels. For example at the method level, message slicing technique is needed and should allow recursive slicing. Recursive slicing is a set of graphs that need further slicing in order to achieve more refined results.

5. Conclusion

These papers more focus on visualization because this paper is continued with change impact research by PhD Student. In this paper, we study a mechanism to handle the impact analysis and visualization in software system. We cater into our work a capability to support the ripple effects with some degree of visualization. Visualization is applicable not only to a proposed change artifact but also to any other artifacts of interest from which we can visualize its dependencies in software system. This requires special attention on the software impact analysis and program structures.

6. References


