An evaluation of four reverse engineering tools for C++ applications

Tung Doan

University of Tampere
Department of Computer Sciences
Computer Sciences
M. Sc. thesis
Supervisor: Eleni Berki
October 2008

University of Tampere
Department of Computer Sciences
Computer Sciences
Tung Doan: An evaluation of four reverse engineering tools for C++ applications
M. Sc. thesis, 75 pages + 1 appendix
October 2008

Abstract

By using reverse engineering tools, the software developer is able to generate the structure of a software system in graphical reports such as hierarchy trees, call graphs, flow charts, class diagrams and then export reports into various formats such as HTML, XML, XMI (Xml Metadata Interchange) or the formats of other reverse engineering tools. C++ programming language supports object-oriented programming and there are more reverse engineering tools supporting this language than other languages such as C# and Java. However, there have been a few evaluation works in comparing, contrasting and thoroughly identifying the capabilities of reverse engineering tools for C++ applications. Therefore, in this thesis, four widely used reverse engineering tools which support C++ are chosen to be examined, namely Rigi, Columbus/CAN, Imagix 4D, and Understand. The tools are evaluated by using them to examine two different types of C++ application: a small game and a large library written in Visual C++. After evaluating them and considering other related research work on evaluation, I outline and comment on the features and capabilities of the tools, along with their strengths and limitations. Last but not least, I provide the reader with some suggestions for designing and implementing an efficient reverse engineering tool for C++ applications.

Keywords: Reverse Engineering, C++, CASE tools, Software quality management, Software maintenance, Reusability.

Table of contents

Figures

Tables

Acknowledgements

I would like to thank my supervisor, Dr. Eleni Berki, at the Department of Computer Sciences for her advice during my work in doing this thesis . Her patience, enthusiasm, responsibility and meaningful guidance helped me very much in directing my thesis to the final destination.

I would also like to thank my girlfriend, Giang Nguyen, for her encouragement and language review as well.

Tampere, October 2008

Tung Doan

1. Introduction

1.1. Overview

Software maintenance is the last phase in the life cycle of a software development process which often includes the following phases: requirements specification, design, implementation, testing, deployment and maintenance. However, this phase plays an important role because software maintenance activities ensure that a software system still works well without errors or in new environments after released. Common maintenance activities include fixing bugs, adapting the system to new environments, adding new features for the system to satisfy new requirements from the client, and updating documentation for the system. In order to do the above tasks, software maintainers must understand the structure or architecture of the system. However, it is a hard task for them while there were some changes in the structure of the system which makes the system different from its original version. The documentation of the system is not up-to-date so it cannot provide explicit knowledge about the system. Source code is the most important available source to understand the structure of the system.

In the case of code reuse, if some parts of a new software system can be reused from existing systems, software developers will save a large amount of money and effort in developing it. Nowadays, there are a large number of open source communities with a lot of open source software systems which are free for you to reuse in developing your new systems. Therefore, when developing a new system, software developers often find out similar open source systems and try to reuse some possible parts of them. However, in order to reuse the source code from the open source systems, software developers must realize their structure and architecture and then understand clearly their features and functions. Unfortunately, it is also a hard task because open source systems are developed and contributed by developers from all over the world so they cannot be managed strictly. As a result, documentation is not produced. Software developers must analyze the source code in order to understand the structure of these systems.

Reverse engineering tools are very useful in the above two cases. They help software maintainers and developers understand the structure of a software system by analyzing the source code and then presenting the system at higher levels of abstractions

such as call graphs, flowcharts, and class diagrams. Like other CASE (Computer Aid Software Engineering) tools, they are useful in handling large and complex projects which are difficult for software engineers to reverse manually [Müller et al., 2000]. In addition, they provide several capabilities such as (i) generating the structure of a software system in graphical reports such as hierarchy trees, call graphs, flow charts, class diagrams; (ii) exporting these reports into various formats such as HTML, XML, XMI or the formats of other reverse engineering tools; (iii) analyzing the software system in these graphical reports, (iv) editing source code in the browser/editor of the tool; and (v) tracking software quality by using software metrics integrated in the tools.

1.2. Research problems

Reverse engineering tools are very useful but they are now not widely used by software engineers [Müller et al., 2000]. There are some reasons of that. For example, available reverse engineering tools are not integrated in popular tools such as IDEs (Integrated Development Environments), unit-testing tools, and code debugging tools which are used widely by software engineers and especially they lack many necessary features and capabilities. However, there were a few works which are relevant to evaluating the capabilities of revere engineering tools for C++. The work of Berndt Bellay and Harald Gall [Bellay and Gall, 1998] in evaluating the four tools: Rigi, Refine/C, Imagix 4D and SNiFF+ is valuable but they only evaluate these tools with embedded systems written by C and Assembly. They did not evaluate other types of software systems written by C++ object-oriented programming language. Especially, their work was done in 1998. It was ten years ago and recently there have been many changes in these tools with many new versions which have been released. For example, the version of Imagix 4D which they evaluated at that time is 2.7 but the newest version of Imagix 4D is 6.3 [Imagix 4D webpage, 2008].

Árpád Beszédes et al. [Beszédes et al., 1999] evaluate only one tool, Columbus/CAN. Comlubus/CAN is an efficient tool for reversing large C++ and visual C++ applications also. Their work is also very valuable when it provided a complete evaluation about all features and capabilities of this tool but they did not compare this tool with other similar tools. Additionally, there have been also many changes in this tool from the time when they evaluate to now.

Storey et al. [Storey et al., 1997] in their article proposed an evaluation on only two types of representation in Rigi: multiple windows and SHriMP (Simple Hierarchical Multi- Perspective) views. It is very specific and does not provide a comprehensive view about the reverse engineering tools for C++ application. Storey et al. [Storey et al., 2002] described the applying of a very efficient tool evaluation technique namely collaborative structured demonstration in evaluating reverse engineering tools in a working session at the Eighth working conference on reverse engineering. However, the result is very little because time limitation. They provided only some short sentences about the parser of Rigi and Columbus/CAN.

1.3. Research questions

The main purpose of my thesis, namely "An evaluation of four reverse engineering tools for C++ applications", is to evaluate the latest versions of four reverse engineering tools for C++ applications which are popular in use: Rigi (version 5.4.1), Columbus/CAN (version 3.5), Imagix 4D (version 6.2.2), Understand (version 2.0) . My motivation of the thesis work is to gain deep understanding in the field of reverse engineering and especially to know the features and capabilities of available reverse engineering tools which are used widely. In my opinion, experience which I have been obtained from the thesis work helps me very much in the near future when coming back to work in my country. Software developers in my country are required to have good software analysis because we often develop new software systems from free reusing components or libraries. In addition, a software manager should have knowledge about all phases in a software development process. By doing the thesis, I have opportunity to study deeply about these phases. Rigi is a free tool which integrates many technologies in representing the structure of software systems in graphical reports but it has many limitations in its user interface and parser. Columbus/CAN is a free tool for the purpose of academic study. It is an efficient tool for reversing large C++ and Visual C++ applications as well. Imagix 4D and Understand are commercial tools which provide many capabilities such as represent and analyze the structure of software in graphical reports and track software quality with software metrics. Their capabilities are evaluated in five main categories: import/export, analysis, browsing/editing, representation and other capabilities such as extensibility, change analysis, and software metrics integrated in these tools. Criteria are

very important in evaluating the capabilities of tools and affect the result of the work. Therefore, I created a complete, consistent, and ease of understanding set of criteria which help me evaluate all necessary features and capabilities of the tools. After evaluating these tools, conclusions are given, along with their strengths and weaknesses. Furthermore, I provide my suggestions for designing an efficient reverse engineering tool for C++ applications. Therefore, my research questions are:

- What are the features and capabilities of the four reverse engineering tools for C++ applications?
- What are the strengths and weaknesses of the four reverse engineering tools for C++ applications?
- What should be the features and capabilities of an efficient reverse engineering tool for C++ applications?

1.4. Research methods

The types of research which are used in this thesis are evaluation research and comparative research. Evaluation research in computer science is based on creating assessment criteria by which an application or a software system can be evaluated for such qualities as effectiveness, validity and ease to use. Comparative research in computer science is based on making comparisons between applications, tools, or software systems. This type of research is used to compare the features and capabilities of the four reverse engineering tools and then to find out the strengths and weaknesses of each tool.

There are many techniques that can be efficient in evaluating reverse engineering tools such as expert reviews, user studies, field observations, case studies and surveys [Müller et al., 2000]. In this thesis, the case studies technique is chosen. This technique means we apply a tool to specific systems [Müller et al., 2000]. In my thesis, I create a case study which includes various kinds of C++ applications such as small pure C++ applications, Visual C++ applications and large applications with hundreds of thousands of lines of source code and then apply the four reverse engineering tools to these applications.

1.5. Outline of the thesis

My thesis is laid out in six chapters. In the next chapter (chapter 2), a background knowledge about reverse engineering is mentioned. They are definition of reverse engineering, sub-areas of reverse engineering, and objectives of reverse engineering. Chapter 3 presents a literature review about the previous studies of evaluating reverse engineering tools for C++ applications: What the other researchers have done and the strengths and weaknesses of their works. Chapter 4 presents an evaluation work of four chosen reverse engineering tools: Rigi, Columbus/CAN, Imagix 4D and Understand. In this chapter, I first describe general information about these tools, along with their features and capabilities, and then create a set of criteria and a case study in order to evaluate the capabilities of these tools. Chapter 5 presents my own evaluation about the strengths and weaknesses of the four tools and then propose some suggestions for creating an efficient reverse engineering tool. Conclusions are discussed in the last chapter (chapter 6).

2. Background

2.1. Definition of reverse engineering of software

Reverse engineering in software engineering is the opposite of forward engineering which is offered to indicate a traditional software development process [Nelson, 1996]. The traditional software development process often includes four phases: analysis, design, implementing, and testing. Through those phases, software is developed from the high level of abstraction (architecture) to the low level of abstraction (source code). Therefore, reverse engineering is the process which analyzes a software system and then represents it at the higher levels of abstraction. The following definition which is given by Chikofsky and Cross II [Chikofski et al., 1990] is widely used:

> Reverse engineering is the process of analyzing a subject system to
>
> • Identify the system's components and their interrelationships and
>
> • Create representations of the system in another form or at a higher level of abstraction.

In order to understand clearly about the reverse engineering term, I compare this term with other terms: *restructuring* and *reengineering*.

Restructuring is the transformation from one representation form to another form within a same abstraction level [Chikofski et al., 1990]. For example, modify source code in order to make the structure of source code more clear. This process only takes place in one abstraction level and its result is the representation of the system in another form depending on the purpose of software engineers but still in the same abstraction level, while reverse engineering deals with many abstraction levels and its result is the representation of the system at a higher level of abstraction. In addition, restructuring creates changes in the structure of the system, while reverse engineering only examines the structure of system and does not make any changes in the system.

Reengineering is the examination, alternation and modification of the system in order to recreate a new system with new functions in another representation form [Chikofski et al., 1990]. This term is wider than the reverse engineering term because it often includes both reverse engineering and forward engineering. The first phase in the reengineering process is using reverse engineering to understand the structure of the old system and represent it at the higher level of abstraction. At that time, some changes are

created at any level of abstraction. The second phase is developing the new system based on the new requirements or functions which have just been recently created. This phase follows steps in forward engineering. Hence, reengineering creates a new system with different features and functionalities from an old system, while reverse engineering does not make any changes in the features and functionalities of the system. Reverse engineering is a process of examination, not a process of modification or replication.

2.2. Sub-areas of reverse engineering

Reverse engineering is a wide area as we mentioned above. Hence, there are many sub-areas in reverse engineering. Two most common sub-areas are *re-documentation* and *design recovery*.

Re-documentation is the creation or revision of another semantic representation within the same abstraction level [Chikofski et al., 1990]. The results of this process are often diagrams reflecting dataflow, code structure or control flow. Re-documentation is considered as the simplest and weakest form of reverse engineering. It aims to provide an easier and clearer way to recognize and understand relationships among all components of the system.

Design recovery is the recreation of a design abstraction representation from not only source code but also other sources such as existing documents, domain and application knowledge, personal experience [Chikofski et al., 1990]. It aims to help software engineers understand fully what the system is, how it works, and why.

Below is the figure which explains all above terms in an abstraction way.

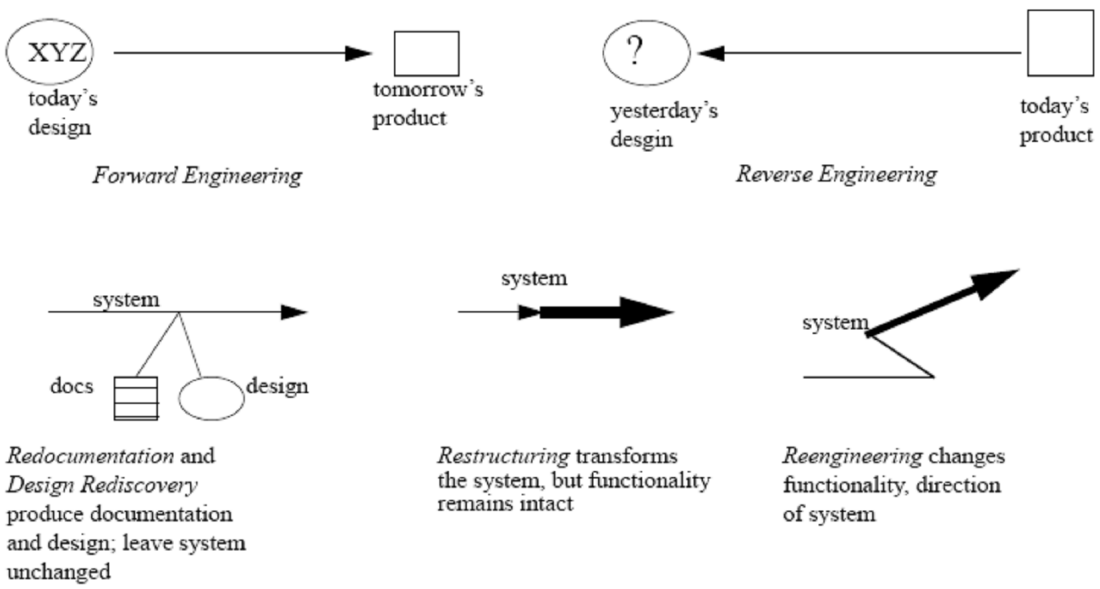

Figure 1: Forward Engineering, Reverse Engineering and Derivatives [Nelson, 1996]

2.3. Objectives of reverse engineering

Software is developed from the highest level of abstraction (architecture) to the lowest level of abstraction (source code). It is not easy to understand the structure of the system in the lowest level of abstraction and only software engineers who have skills and experiences in programming could understand it. The system at higher levels of abstraction is more understandable. Hence, using reverse engineering tools and methods is one of the most efficient ways to understand clearly the structure of the system if we only have its source code. When software developers are required to create a new system, they first often find similar systems which have some same features and functionalities with the requirements for their new system. By using reverse engineering tools to examine these systems, they can find reuse functions or components which could be used for their system. In addition, reverse engineering is very useful for software maintenance. After a long time of being used and modified, the system probably has errors and needs to be maintained. However, there have been some changes in the structure of the system and it probably is different with the original version. Using reverse engineering to create higher levels of abstraction of the current system could help software maintainers understand clearly about the system. There are six key objectives of reverse engineering given by Chikofsky and Cross II [Chikofski et al., 1990]: *cope with complexity, generate*

alternative views, recover lost information, detect side effects, synthesize high abstractions, and facilitate reuse.

Cope with complexity: Using reverse engineering tools, which can examine and analyze automatically thousands of lines of source code and then present relationships among the system's components and the structure of the system, is an efficient way to deal with large and complex software systems.

Generate alternative views: Graphical representations are easier for software engineers to understand the structure of the system. Possible alternative views are graphs, structure charts, data flow diagrams, control flow diagrams, class diagrams, and entity-relationship diagrams.

Recover lost information: After being used for a long period, the system is different from its original version. There have been some changes in its source code and hence its old documents are not up-to-date. Reverse engineering can recover this lost information and represent the system at higher levels of abstraction.

Detect side effects (ramifications): In the implementation phase, software developers implement the system based on the design documents of the system. However, in some cases, they make some changes in the structure of the system when implementing it. This will create some ramifications of the system and hence the structure of the real system is not same as the structure of the system in the design documents. Reverse engineering process analyzes the source code and therefore will detect these ramifications.

Synthesize high abstractions: The main nature of reverse engineering is representing the system at higher abstractions. There are various levels of abstraction which are created and reverse engineering process can synthesizes them. This helps software engineers easy to understand the structure of the system.

Facilitate reuse: Reverse engineering examines the structure of the system in order to find out possible reusing components. Using these reusing components to develop new systems will save a lot of effort, cost and time.

2.4. The generic process of a reverse engineering

According to Scott Tilley [Tilley, 1998], the reverse engineering process includes three main activities: *data gathering, knowledge management and information exploration* .

Data gathering

We cannot understand about the structure of a software system at higher levels of abstraction if we do not have information about it. Therefore, data gathering is often the first step in the reverse engineering process. In this step, many types of data about the system are gathered such as source code, comments in source code, documents about the system, and experience of experts. Three techniques of data gathering which are widely used are system examination, document scanning and experience capture [Tilley, 1998].

System examination is often classified into two contrasting ways: static examination and dynamic examination. The static examination concentrates on anal yzing the source code. A source code parser is often used to analyze the source code and then transfer it to abstract syntax trees [Bellay and Gall, 1998]. In contract, the dynamic examination focuses on the executing system. It is useful for understanding component - based systems in which the static examination cannot apply because components do not come with the source code. Analyzing systems when they are running helps us to have knowledge about the interactions between components in the system, types of messages and protocols used and the external resources used by the system [Tilley, 1998]. Distributed, real-time and client-server applications are analyzed efficiently by this technique.

Document scanning is the process of gathering documents, another type of information about the system. For example, comments in the source code are useful sources for understanding the system. However, automatic analysis of the comments is more difficult as they may be isolated in the source code or they do not provide explicit information about the source code when they are not updated. Therefore, comments are often analyzed manually by experts.

Experience capture is the approach to obtain knowledge about the system by interviewing with people who has developed the system. The knowledge is very useful for understanding the system. However, it is difficult to find out who developed the sy stem.

Knowledge management

Knowledge management in reverse engineering is used to structure gathered data into a conceptual model of the application domain called a domain model. It includes three main steps: knowledge organization, knowledge discovery and knowledge evolution [Tilley, 1998].

Knowledge organization describes mechanisms to structure gathered data into a form called a data model which helps us understand the properties of artifacts and their interrelationships in a software system [Tilley, 1998]. A data model captures both the static and dynamic properties of the system. Static properties are objects, their relationships and their attributes. Dynamic properties are operations on objects, their relationships and their properties.

Knowledge discovery describes techniques which are used to support information exploration [Tilley, 1998]. For example, some of the techniques are navigating, structuring and visualizing the knowledge base that is a collection of objects and their associations.

Knowledge evolution describes the way in which knowledge is updated during the reverse engineering process [Tilley, 1998]. Iterative domain modeling is one form of knowledge evolution. During the reverse engineering process, software engineers recognize the components and their relationships and then update to the domain model.

Information exploration

The process of information exploration includes three activities: navigation, analysis and presentation [Tilley, 1998].

Navigation is the ability to select, edit and traverse artifacts in the information space. Artifacts are selected based on their attributes, their properties, visual and spatial cues or other criteria. The user also can edit artifacts or their relationships such as add new artifacts, modify their attributes and properties, or delete existing ones. Finally, the user can move from one artifact to another one.

Analysis is the activity of analyzing the software system which uses various techniques such as static analysis, dynamic analysis and impact analysis. Analysis capabilities are possible in many levels of abstraction such as call graphs, flow charts or

class diagrams. We should pay attention to how to balance between automatic, semi-automatic and manual analysis in order to get high efficiency.

Presentation is the activity to represent the structure of the system at high levels of abstraction. The user interface of reverse engineering tools provide many views to represent the structure of the system at various levels of abstraction such as call graphs, flow charts, control flows, class diagrams and hierarchical graphs. Advanced visualization techniques are used such as fish-eye views and three-dimensional imaging.

2.5. Reverse engineering methods and techniques

Several techniques have been invented and widely used for supporting reverse engineering activities. According to Tonella et al. [Tonella et al., 2007] and Müller et al. [Müller et al., 2000], such methods and techniques include code visualization, program slicing, concept/feature location, design recovery, dependency analysis, clustering, clone detection, and impact analysis. In the following, brief descriptions of each technique or method are given.

Code visualization: This technique uses typography, graphic design or animation to provide the ability to comprehend the large amount of source code [Lanza, 2003]. It can visualize both static information (the structure of a system) and dynamic information (the executing system). The process of code visualization in a visualization tool often includes three steps: gathering and storing of source code in data models, handling gathered data and representing it in internal representations, and representing the output in graphical views.

Program slicing: Program slicing is a technique to determine the parts of a program which affect the values computed by a particular slicing criterion [Tip, 1994]. This is done by deleting other parts which do not affect the values computed by this criterion. There are two types of slicing: static and dynamic. In static method, only statically available information is used for computing slices whereas in dynamic method, the input is specified before the computing.

Concept/feature location: This method provides the ability to isolate some parts of code which are responsible for the implementation of a given concept or feature [Tonella et al., 2007].

Design recovery: This method aims to discover the structure or architecture of a software system. In order to do this, we need information about the system not only so urce code but also other types of information such as documents and experts' experience.

Dependency analysis: This techniques aims to discover the dependency of software artifacts [Systä, 1999]. In reverse engineering, software artifacts are often represen ted in a dependency graph which helps software developers analyze the dependency of the system easily.

Clustering: This technique aims to group related methods, variables. For example, all methods use a specific variable or all methods call a specific meth od [Quigley et al., 2000].

Clone detection: This technique is used to detect code elements which were replicated by comparing the abstract syntax trees, the metrics or the descriptions of program elements [Tonella et al., 2007].

Impact analysis: This technique aims to estimate the effect of changes on the system. For example, analyze how the system will be changed if there are some changes in some particular parts. In reverse engineering, this technique is used in analyzing the system at syntactic level [Tilley, 1998]. In addition, this technique is only useful when the traceability of the system is high.

2.6. Challenges

The main challenge in reverse engineering is how to store and analyze information about a software system at various levels of abstrac tion, not only source code and then provide traceability of software artifacts. The source code often does not contain all information about the system. There is still much knowledge which is very important to understand about not only the structure of the system but also the evolution of the system such as business plan, application domain, architecture description and engineering constraints [Müller et al., 2000]. Therefore, it is necessary to create models that capture, stores and handle all information about the system at various levels of abstraction in a consistent way and to provide an efficient traceability. For example, given a design module, it is able to point out the code elements that implement it, the functional specification elements in the requirement specification and other corresponding elements in the other levels of abstraction.

3. Literature review

3.1 Introduction

In this chapter, I describe the literature review work from considering other related works. I have found only four articles which are closely relevant to evaluating the capabilities of reverse engineering tools for C++ applications. The first article provides an evaluation of the four reverse engineering tools: Refine/C, Imagix 4D, SNiFF+, and Rigi [Bellay and Gall, 1998]. The second article describes a working session at the Eighth working conference on reverse engineering which is arranged to evaluate the capabilities of some reverse engineering tools such as Rigi, Columbus/CAN and CodeCrawler [Storey et al., 2002]. The third article provides an evaluation of only one tool, namely Columbus/CAN by using it to examine three different types of C++ projects [Beszédes et al., 1999]. The last article reports the evaluation work of two types of representation: Multiple windows and SHriMP (Simple Hierarchical Multi-Perspective) views) of the Rigi tool [Storey et al., 1997].

3.2 Techniques in evaluating the capabilities of reverse engineering tools

There are several investigative techniques and empirical studies which are useful for evaluating the capabilities of reverse engineering tools such as expert reviews, user studies, field observations, case studies and surveys [Müller et al., 2000]. The authors of the above articles used one technique or a combination of many techniques.

Berndt Bellay and Harald Gall, two researchers from Technical University of Vienna, Austria who proposed a complete and systematic an evaluation of reverse engineering tools in their article "An evaluation of reverse eng ineering tool capabilities", used the case studies technique [Bellay and Gall, 1998]. They used a real-world embedded software system which is a part of the Train Control System as a case study. This system is written by C and Assembler languages with approximate 150K LOC (Line of Code) in total. The quality of source code is quite good with a lot of comments embedded in source code. The documentation of this system is also available for their work. The main purpose of their work is to evaluate the capabilities of the four tools in reversing embedded software systems hence their case study is efficient for their work. In

addition, a system with 150K LOC is enough complexity for evaluating the usefulness of these tools.

Árpád Beszédes et al., who proposed an article which describes their work from evaluating the capabilities of the Columbus/CAN reverse engineering tool, used the case studies technique [Beszédes et al., 1999]. In their case study, three different types of applications are chosen. The first one is a large C++ application consisting of about two hundreds source files with only normal classes. The second one is partial application of six files with complicated templates. The last one is an application made by MFC library. Three different types of application make their work more convincing because they tests the capabilities of the tool with both small applications and large applications, along with Visual C++ applications.

Storey et al., who proposed an evaluation on two types of representation in Rigi: multiple windows and SHriMP (Simple Hierarchical Multi-Perspective) views, used the user studies technique [Storey et al., 1997]. The authors made a test for a group of twelve members. They are required to perform some tasks in using two types of representation in Rigi and then answer some questions and take an informal interview.

Storey et al. in the article, namely "A collaborative demonstration of reverse engineering tools" described the applying of a tool evaluation technique namely collaborative structured demonstration in evaluating reverse engineering tools in a working session at the Eighth working conference on reverse engineering [Storey et al., 2002]. Collaborative demonstration is a technique which evaluates tools by combining various elements such as experiments, case studies, technology demonstration and benchmarking. In the context of the project proposed in this article, there were six teams participating in reversing a system which consists of approximately 30 KLOC of C++ code and then proposing an architecture that is up-to-date with changes made during its evolution. One of the three main goals of the project was to evaluate reverse engineering tools by comparing them and develop better ones. The teams were expected to use different tools, techniques to reverse this system and each team collaborated and used the results from other teams. Each team had specific tasks. The first team (KBGE group) developed a parsing tool and employs approaches to clustering, using hierarchical algorithms. The second team (RGAI) used CAN/Columbus parser/analyzer to analyze

source code. The third team (Rigi) used Rigi C++ parser and TkSee C++ parser to analyze source code and use Rigi graph editor to visualize the system. The fourth team (SWAG) used CPPX parser to analyze source code of the system . The fifth team (SCG) used SniFF++, Moose, and CodeCrawler tools. The last team used GraphTool for visualizing the GXL code generated by TkSee C++.

Collaborative demonstration is an efficient technique because it evaluates tools by combining various elements such as experiments, case studies, technology demonstration and benchmarking. However, it requires much effort and time. The case studies technique is the best choice for evaluating tools by one or a small group of people. The user studies technique is useful because it evaluate tools by various types of knowledge from the user.

3.3. Assessment criteria for evaluating reverse engineering tools

Berndt Bellay and Harald Gall defined a set of assessment criteria in four main categories: analysis, representation, editing/browsing and general capabilities to evaluate the capabilities of the above four tools [Bellay and Gall, 1998]. The analysis category includes criteria used to evaluate the parser of each tool. They are divided into three sub-categories: source types and project definition, parser functionality and parsing functionality. There are four criteria in the first sub-categories: parsable source languages (which source code can be parsed), other importable sources, project definition types and ease of project definition. The second one includes six criteria: incremental parsing, reparsing, fault-tolerant parser (ability to parse incomplete and in correct code), define and undefined, preprocessor command configurable, and support for additional compiler switches. The third one consists of five criteria: quality of parse error statements, parse abortable, point and click movement form parse results to source code, parsing results and parse speed. The representation category includes criteria which are used to evaluate the properties and quality of reports such as: speed of generation, filters, scopes, grouping (ability to present only the part of the graph) and navigating between reports and between a report and source code. They divided reports into two kinds: textual one and graphic one. There is only one criterion used to evaluate the textual one. It is the sorting capability. Criteria used to evaluate the graphic one are layout algorithms, view editable, layered view and SHriMP (Simple Heirachical Multi Perspective) views. The editing/browsing category includes following criteria: intelligent control of text

editor/browser, highlighting of the source code, search function, hypertext capabilities, and a history of the browsed locations. The last category includes five criteria in evaluating general capabilities: supported platforms, multi-user support, toolset extensibility, storing capabilities, and output capabilities. In addition, they used three methods to assess the quality of each tool in each criterion: an enumeration of possible types, yes or no (the availability of a capability or not), and a simple four-level scale (excellent, good, acceptable, and not at all).

Árpád Beszédes et al. also defined assessment criteria for their work from evaluating Columbus/CAN [Beszédes et al., 1999]. Their criteria are based on the criteria of Bellay and Gall. They are assigned into five categories: analysis (the capability of source code parser), import/export (the capability of importing existing projects and exporting to various formats of presentation), representation (the capability of representing the results of the parsing process), editing/browsing (the capability of editors/browsers), and general capabilities (user interface, extensibility, storing capability, multi-user support, among other things).

Criteria defined by Berndt Bellay and Harald Gall, are quite complete since it covers all aspects of a reverse engineering tool from main features and capabilities (parsing, representation, browsing/editing) to small features and capabilities (import/export, the quality of error statements in parsing, search function). The criteria are sorted and classified clearly into big categories and the categories include sub-categories.

3.4. Results from the previous studies in evaluating reverse engineering tools

Berndt Bellay and Harald Gall provided their conclusions about the four tools: Rigi, Refine/C, Imagix 4D, and SNiFF+ [Bellay and Gall, 1998]. The analysis capabilities of Refine/C are excellent as the results of the parsing process are exact. This is the reason why it is widely used in the reverse engineering community. It also provides several capabilities in parsing such as define and undefined per file, exclusion of files. However, one main limitation of the parser of Refine/C is do not support reparsing. The user interface of Refine/C is not highly appreciated. It provides only one representation of each view and the user can not change the position of these representations. Especially, it does not have an integrated editor and support search engine. There are also limitations in

representation capabilities (only the movement of the entities and browsing through the view is supported). About its extensibility, it provides an API which permits developers access to its features to build customized analysis tools and to C parser and printer to enable extensions to grammar, lexical analyzer. Imagix 4D also provides an excellent parser which supports reparsing, incremental parsing. The project definition capabi lity is flexible when the user can define a project by file, directory, or makefile. It also supports the import of additional data sources (graph profile data -gprof and test coverage data - tcov). The user interface of Imagix 4D is friendly, easy to use and efficient with several features and capabilities. For example, there are several representations in each view, its integrated editor is efficient with highlighting and movement in the editor capabilities, the search capability is quite good, and it provid es filtering, scoping and grouping capabilities to help the user can narrow the view of the whole system and see some parts of the complete system. One main strength of Imagix 4D is the capability to automatically generate documents from the source code. H owever, it also has two main weaknesses: extensibility and the generation of graphical views in printed form. Rigi does not have an efficient parser when it only parses functions and struct data type. The user interface is not friendly and do not provide several capabilities which Imagix 4D does. The main advantages of this tool are some new features which do not occur in other tools such as layered views, SHriMP view and layout algorithms. SNiFF+ provides a fast and tolerant parser. It means that it can pa rse source codes which are incomplete and incorrect. The user interface of SNiFF+ provides an integrated editor which is as good as the Imagix 4D's editor. The view of SNiFF++ is suitable for printing but not for comprehension. One main limitation of this tool is that it can not be extensible.

The results from evaluating the tools in the working session with the collaborative demonstration technique were gathered from reports of the teams. The results proposed in the article are very few [Storey et al., 2002]. In comparing parsers (CAN/Columbus, CPPX, Rigi and TkSee/SN), the authors concluded that they made different in their level of detail and output formats: Can/Columbus and CPPX emit facts at the AST level, Rigi emits RSF (Rigi Stadard Format), TkSee emi ts GXL (the emerging standard format for exchanging data between reverse engineering tools) at the middle level (external declaration level). The authors did not mention about the speed and effectiveness of these

parsers. In evaluating documenting and visualizing capabilities, the authors only provided snapshots when using these tools (CodeCrawler, GraphTool, PBS, and Rigi) but did not propose any analysis about them.

The results of the evaluation of Columbus/CAN are given in the last section of the third article [Beszédes et al., 1999]. In terms of analysis capabilities, the user can handle different programming languages in a single project. Its parser also works well while it can recognize all C/C++ types, namespaces, nested classes, templates. The relationship among the objects of the system is parsed well such as inheritance, aggregation, general association. The parser can support fault-tolerant, re-extraction but does not support incremental parsing. In terms of import/export capabilities, it can import MS Visual C++ 6.0 projects, There are three options for the user when exporting: exporting into an ASCII file, into a MS Jet Database or into a TDE repository. The user can add new exporters using the exporter/ extractor API. In addition, TDE/Columbus can create documents in SGML or HTML formats. In terms of representation, the final output is represented in form of UML diagrams. The user can use filtering capability to see some particular parts of the whole diagram. The user can filter according to scopes/namespaces, using class dependencies or manually. The dynamic view is not supported. It means that the user cannot switch between the representation and the source code. In terms of editing/browsing, there is not a text editor in Columbus but the user can use any external text editor because TDE acts as an OLE client, and any text editor acts as an OLE server can access it. In terms of general capabilities, this tool is easy to extend using the APIs, and supports multi-user.

The results from evaluating two types of representation in Rigi suggested that the user was more satisfied with SHriMP approach than multiple windows [Storey et al., 1997].

3.5. Conclusions

There were a few works which are relevant to evaluating the capabilities of revere engineering tools for C++.

The work of Berndt Bellay and Harald Gall in evaluating the four tools: Rigi, Refine/C, Imagix 4D and SNiFF+ is valuable but they only evaluate these tools with embedded systems written by C and Assembly. They did not evaluate other types of

software systems written by C++ object-oriented programming language. Especially, their work was done in 1998. It was ten year ago and now, there have been many changes in these tools with many new versions which have been released. For example, the version of Imagix 4D which they evaluated at that time is 2.7 but now, the newest version of Imagix 4D is 6.3 [Imagix 4D webpage, 2008].

Árpád Beszédes et al. evaluate only one tool, Columbus/CAN. Comlubus/CAN is an efficient tool for reversing large C++ and visual C++ applications also. Their work is also very valuable when it provided a complete evaluation about all features and capabilities of this tool but they did not compare this tool with other similar tools. One more thing, there have been also many changes in this tool from the time when they evaluate to now.

Storey et al. in their article proposed an evaluation on only two types of representation in Rigi: Multiple windows and SHriMP (Simple Hierarchical Multi-Perspective) views. It is very specific and does not provide general view about the reverse engineering tools for C++ applications.

Storey et al. described the applying of a very efficient tool evaluation technique namely collaborative structured demonstration in evaluating reverse engineering tools in a working session at the Eighth working conference on reverse engineering. However, the result is very little because of the limitations of time. They provided only some short sentences about the parser of Rigi, Columbus/CAN.

4. An evaluation of the capabilities of the four tools

4.1. Overview of tools

Rigi

Rigi is a free reverse engineering tool for understanding legacy systems developed by a research group in the Department of Computer Science at the University of Victoria, Canada [Rigi webpage, 2008]. This tool aims to discover higher levels of abstraction of software systems for maintenance and evolution purposes. It includes three main components: a parsing subsystem, a repository, and an interactive graph editor [Müller et al., 1993]. The parsing subsystem now supports C, C+ and COBOL languages. The repository stores the results of parsing process. It supports multi –user and distributed use. The graph editor is called "rigiedit" which provides browsing, editing, manipulating, exploring and managing capabilities [Bellay and Gall, 1998]. The user can view parts of the whole graph by using filters and can also edit the graph through rigiedit. Rigi runs on several platforms such as Windows, Linux and Solaris.

Columbus/CAN

Columbus/CAN is a commercial reverse engineering tool developed in corporation between the Research Group on Artificial Intelligence in Szeged, the Software Technology Laboratory of the Nokia Research Center and FrontEndART Ltd [Ferenc et al., 2002], but it is free for the user who would like to use it for academic and educational purposes. It aims to parse, analyze, filter, and export information embedded in C/C++ source files into various kinds of formats such as ASCII, HTML, and XML [Beszédes et al., 1999]. This tool comprises a friendly user–interface that looks like integrated development environments (IDEs) which combines various reverse engineering tasks such as project handling, data extraction, data representation, data storage, filtering, and visualization [Beszédes et al., 2005]; a powerful parser which supports incremental and fault–tolerant parsing, handling of templates; and a database. It runs on Windows platform.

Imagix 4D

Imagix 4D is a commercial reverse engineering tool released by Imagix Corporation [Imagix 4D webpage, 2008]. By using this tool, software engineers can speed their work

in developing, reusing, testing and maintaining software systems because it is an efficient tool for checking rapidly and systematically the structure of such systems at an y level of abstraction, analyzing flow charts and control flow, tracking the quality of the software system by metrics, and generating automatically documents. The architecture of this tool comprises three main layers: a view, an exploration engine, and a database [Imagix 4D webpage, 2008]. The view is where you handle your tasks, see and manipulate the results such as UML diagrams, flow charts, or software metrics, etc. The exploration engine is a machine to receive requests from the users through the view layer, access the database to handle requests and then send answers to the users in the view layer. The database stores information about the software system such as source code, makefiles, profile results, etc. You can use this tool for reversing softwar e systems written by C, C++ and Java languages. It can run on several platforms such as Windows, Linux, and Solaris.

Understand

Understand is a cross-platform, multi-language reverse engineering tool developed by Scientific Toolworks company [Understand webpage, 2008]. It can analyze source code written by one of nine programming languages such as C/C++, C#, Java, Pascal and run on several platforms such as Windows, Linux, and Solaris. Especially, it can analyze source code written by various programming l anguages at the same time in a single project. Its IDE (interactive development environment) is very flexible. The user can create their own workplace to organize windows which view different information such as source code, metrics, graphs, charts. In add ition, it offers several functionalities such as several graphical reverse engineering views, code navigation using a detailed cross reference, a syntax colorizing editor, a lot of metrics [Understand webpage, 2008].

4.2. The features and functionalities o f tools

Rigi

Extensibility: The user can easily extend the core functionalities of Rigi as it does not provide fixed numbers of techniques for data gathering, analysis, organization and representation. It is flexible for the user to choose suitable techniq ues for their needs. It also enables the user to interpolate with other tools in an integrated way to extend its functionalities [Rigi webpage, 2008].

Customization: Rigi enables the user to personalize the user interface. In addition, the architecture of Rigi is based on a domain-retargetable approach, hence the user can model application domain [Rigi webpage, 2008].

Representation: Rigi proposes two contrasting approaches for presenting the structure of software system: multiple windows and SHriMP (Simple Hierarchical Multi-Perspective) views [Storey et al., 1997]. The structure of software systems is often presented by a hierarchy graph with nodes representing system artifacts such as functions, datatypes and arcs representing the relationships of artifacts. In the case of the first approach, this hierarchy is represented by individual and overlapping windows. Each window displays a specific slice of hierarchy. With this approach, the user cannot see the whole structure of software system. Therefore, it is not efficient for software systems with a large structure. In contrast, with the second approach, the user can see the whole structure of the software system in a nested graph. The algorithm used in this approach is a fisheye view. The fish eye view means that you can see simultaneously the local detail and global context of a graph.

Other features:

- Evaluates the precise dependences between two subsystems and the impact of a change to the source code.
- Provides metrics for cohesion and coupling
- Includes a built – in scripting language and command library.
- Adapts to different programming languages [Rigi webpage, 2008].

Columbus/CAN

Extensibility: The architecture of Columbus is based on plug-ins, hence it is easy to extend core functionalities. The user can use an easy-to-use plug-in API to write and add new functionalities into the Columbus system or to connect the system with other tools [Beszédes et al., 1999].

Project handling: Columbus enables the user to import MS Visual C++ and .NET projects, to handle huge projects, or to handle several languages in the same project [Demeyer et al., 1999].

Filtering: Columbus provides four types of filtering as follows [Ferenc et al., 2002]:

- Filtering by input source files: Only classes within given input source files are displayed.
- Filtering according to scopes: The user can choose which will be displayed in classes or namespaces from view-tree browser.
- Filtering using class dependencies (aggregation, inheritance): The user can see all derived classes from a give class.
- Filtering "by hand": The user can select/deselect classes to be showed in the IDE.

Exporting: Columbus exports output in several formats such as CPPML (C++ Markup Language), GXL (Graphic eXchange Language), HTML and especially formats which can be handled by other reverse engineering tools such as Rigi (RSF format), CodeCrawler (Famix XMI format), Maisa [Ferenc et al., 2002].

Imagix 4D

Representation: Imagix 4D exports the output in high level abstractions by providing several abstraction mechanisms: UML class diagrams, UML file diagrams and build-in abstractions [Imagix 4D webpage, 2008]. UML class diagrams help you view and then understand the static structure of software with relationships between classes. You can view just a class with its attributes and operations or several classes with their relationships in a class hierarchy. This helps you understand large and complex software. UML file diagrams display information at the file level such as the location of files, the elements of files, and their build dependences. You also use # directives for improving build times and reuse. One of build-in abstraction mechanisms is grouping. You can choose related classes or methods to form a group.

Browsing: You can see the structure of software at various levels of abstraction within the browser of the Imagix 4D [Imagix 4D webpage, 2008]. A file editor is integrated into the browser hence you can see both class diagrams and source code. Hence, it is easier for you to understand the structure of software. The symbols (classes, functions, types and variables) are color-coded. It also supports source code navigation.

Quality checks: Quality checks help the user to identify potential problems which occur in the run-time execution of their software. It provides capabilities to analyze data flow of source code in order to find out problems about data access, concurrency control. The

user can also review possible conflicts in real-time, embedded, and multi-threaded systems [Imagix 4D webpage, 2008].

Software metrics: Imagix 4D provides more than seventy metrics in order to measure various aspects of software such as quantity, quality, complex, and design of software [Imagix 4D webpage, 2008]. These metrics are mostly divided into four categories corresponding with four levels: file (eighteen metrics), class (seventeen metrics), function (seventeen metrics) and variable (three metrics). Besides common metrics such as lines of code, line of comments, comment ratio, numbers of statements at both file level and whole project level, there are specific metrics such as McCabe cyclomatic complexity and Hastead program difficulty metrics for testability and maintainability purposes or Chidamber and Kemerer metrics to measure the class coupling and class cohesion of object-oriented software. Metrics are displayed on metrics windows and the user can list, sort, rank and compare all symbols based on their attributes.

Document generation: Imagix 4D can automatically generate technical documents from information in source code and Imagix 4D's database [Imagix 4D webpage, 2008]. Hence, you save development effort for writing documents. Moreover, you always have up-to-date documents of your software. Documents are in three formats: ASCII, RTF (rich text format) and HTML.

Understand

Combined language analysis: Understand provides the capability to examine source code written by more than one programming language in a single project [Understand webpage, 2008]. For example, the user is able to reverse Java and C++ source code at the same time in a project. In addition, this tool supports analysis about the dependence of parts of the whole system which are written by various programming languages.

Customized interface: The user interface of Understand is friendly, easy to use and especially looks like an IDE. Furthermore, the user can organize the position of and then create a specific workplace which they want [Understand webpage, 2008]. The user interface provides several windows which include information about the system from various aspects such as architecture browser, project browser, metrics, integrated editor and diagrams.

Change analysis: Understand is an efficient tool for maintaining software systems because of its change analysis capability. The user is able to compare between two files, two folders in order to know which file or folder is changed from the previous version [Understand webpage, 2008]. Additionally, the user can compare between two sections in a file in order to find, for example, why one section of source code run well but another section of source code does not run well.

Metrics: Understand provides the large number of software metrics (approximately sixty eight) which include statistics about various aspects of a software system [Understand webpage, 2008]. These metrics are mostly divided into five categories corresponding with five levels: project, file (number of files, number of header files, number of code files and among other things), function or program unit (number of program units, number of local methods, number of local private methods and among other things), class (number of base classes, number of immediate subclasses, maximum depth of inheritance tree and among other things) and variable (number of instance variables, number of protected instance variables and among other things). In addition, this tool also provides metrics about the cyclomatic complexity of the system.

4.3. Assessment criteria

This section defines criteria to be used for evaluating the above four tools. The criteria are based on my experience in using reverse engineering tools and the criteria defined by Berndt Bellay and Harald Gall [Bellay and Gall, 1998]. They are organized in a clear hierarchy structure including five main categories: import/export, analysis, browsing/editing, representation and other capabilities. Each category often has sub-categories. The criteria in the first four categories are used to evaluate all common features and capabilities of a reverse engineering tool, for instance, importing sources, parsing source code, representing the results of the parsing process, exporting the results and analyzing the results directly in the tools. The criteria in the last category are used to evaluate other important capabilities such as software metrics, extensibility, and supported platforms. The rational of the criteria are as simple as possible but consistent, complete, effective and precise.

4.3.1 Import/Export

The import capability of each reverse engineering tool is significant because importing is often the first task when using such tools. Furthermore, it defines the typ es of data which the tool can import. The input of reverse engineering tools often includes source code. However other types of data such as documents and experts' knowledge are very use ful to examine a software system. With tools which support C++, the ability to import source code written by Visual C++ is also very essential.

The export capability plays an important role in the usability and efficiency of a reverse engineering tool. If a reverse engineering tool provides the ability to export the output to various formats or the formats of other CASE tools , it will be highly graded. This capability helps the software engineer store the results or use them in other tools to gain better results. As a result, the following is some criteria to be considered for the import/export capability of a reverse engineering tool.

Importable source code types: This indicates which programming languages of source code can be imported to parse. We are ev aluating reverse engineering tools for C++ applications, but these tools often support reversing applications written by different programming languages.

Project definition types: This indicates how a project can be defined in the reverse engineering tools. There are three common methods of definition: file, directory, and makefile.

Other importable sources: Some tools can import other sources such as documents in order to get enough information about applications which will be reversed.

Output formats: This lets us know which format of output can be exported by reverse engineering tools. For example, formats can be ASCII, HTML, RTF and among others.

Easy of project definition: Evaluate the ease of project definition.

4.3.2 Analysis

The source code parser is the most important subsystem of every reverse engineering tool. The results of all tasks depend on the result of the parsing process. For example, if the result of the parsing process is incorrect, the structure of the software system will be represented incorrectly. The functionalities of a parser such as reparsing, incremental parsing and fault tolerant parsing are very useful to reduce time and effort for the process.

The parsing speech is also important when parsing large and complex software systems. As a result, the following criteria should be used to evaluate the parser of a reverse engineering tool.

Incremental parsing: Incremental parsing is the capability to parse only some parts of the whole source code which are changed from the last parsing. This capability makes the parsing process reduce the parse time.

Reparsing: During the parsing process, there are always changes and the use of incremental parsing probably does not make a precise result for the whole source code. Hence, reparsing the whole source code is the best way to obtain an accurate result.

Fault tolerant parsing: This is the ability to continue the parsing process when some errors are occurring. It means the ability to parse incorrect or incomplete source code.

Define and undefine: Two types of define and undefine preprocessor commands should be supported by the parser: define and undefine for the whole project or for each file in the project.

Quality of error statements: Error statements are very important for the user to understand where the errors come from and then know how to fix them. Statements should be understandable, clear and precise.

Capability to abort parsing: The abort capability is also very important to cancel the parsing process when it runs without termination.

Parsing results: Estimate the result of parsing. It should be correct, complete, and consistent.

Parsing speed: The speed of parsing is one of the important features to assess a reverse engineering tool, because C++ applications now are huge projects with hundreds of thousand of line of code.

4.3.3 Browsing/Editing

The browser/editor is necessary for every reverse engineering tool because the software developer needs not only importing the source code and then exporting the results but also analyzing the source code or switching between the source code and a high level of abstraction. Therefore, the capabilities of a browser/editor should be evaluated when considering reverse engineering tools.

Integrated text editor/browser: A text editor/browser is necessary to view and edit source code before parsed. It also is necessary for the user to switch between source code level and architecture level.

External editor/browser: Some tools provide external editors/browsers.

Control capabilities of text editor/browser: The efficient control capabilities of text editor/browser are very necessary for the user to handle source code. These capabilities include, among others, positioning at the right place, counting the appreciate position of an element in the editor/browser, opening a file for editing or browsing.

The usability of user interface: The user interface should be friendly and easy to use. It should also look like popular IDEs because the user is always familiar with using IDEs.

Search function: A search function is useful when the user want to find a word in a file with a lot of words. Search function is therefore necessary for most of browsers/editors.

Highlight capability of source code: The highlight capability makes it easy for the user to understand the structure of source code in browsers.

Hypertext capabilities: This is the capability to jump from an element to another element in a file or among files. This capability helps the user to know the relationship between two elements.

4.3.4 Representation

The representation is also play an important role like the source code parser. A reverse engineering tools should provide many graphical views for the user can see the structure of a software system. Dealing with large and complex systems is a challenge for such tools. Therefore, an efficient tool is one which provides techniques and functionalities for representing efficiently the structure of the software system. Moreover, the ability to switch among views is very useful for the user to analyze the system.

Static/dynamic views: Dynamic view means that when there are some changes in the source code, the output report dynamically reflect these changes. This technique is necessary for using incremental parsing. Static view means that the output report only reflects changes inside the source code when users reparse the whole source code.

Layered view: The elements of the output report are viewed in different layers in one window or many windows.

Filtering, scoping and grouping: These techniques are very necessary for the user to narrow entities in the huge output reports. The user can view only some parts of the whole graph in the reports.

Movement between reports: The ability to navigate from a point in one report to another point in another report.

Movement from reports to source code: The ability to switch between levels of abstraction: switching between source code level and architecture level.

4.3.5 Other capabilities

Supported platforms: Tools should run on many platforms in order to attract more users. For example, the user is using a platform and the tool does not support this platform, the user will find another tool instead of removing his/her platform and using another platform.

Integrated metrics: Metrics are important, for example they are used to track the quality of source code during the development process, to estimate the complexity of the application.

Change analysis: The ability to compare files in the directory, texts in the files and other things to realize the difference among various versions.

Extensibility: The architecture of reverse engineering tools should be easy to extend from core architecture and to link with other tools. The ability to link with other tools helps the user to use the strength of each tool and then to get better results.

4.4. Case study

I choose two different types of C++ projects in the case study for evaluating the capabilities of the four tools.

The first project is a small and simple game which includes only five files (.cpp) and four classes with about 190 LOC.

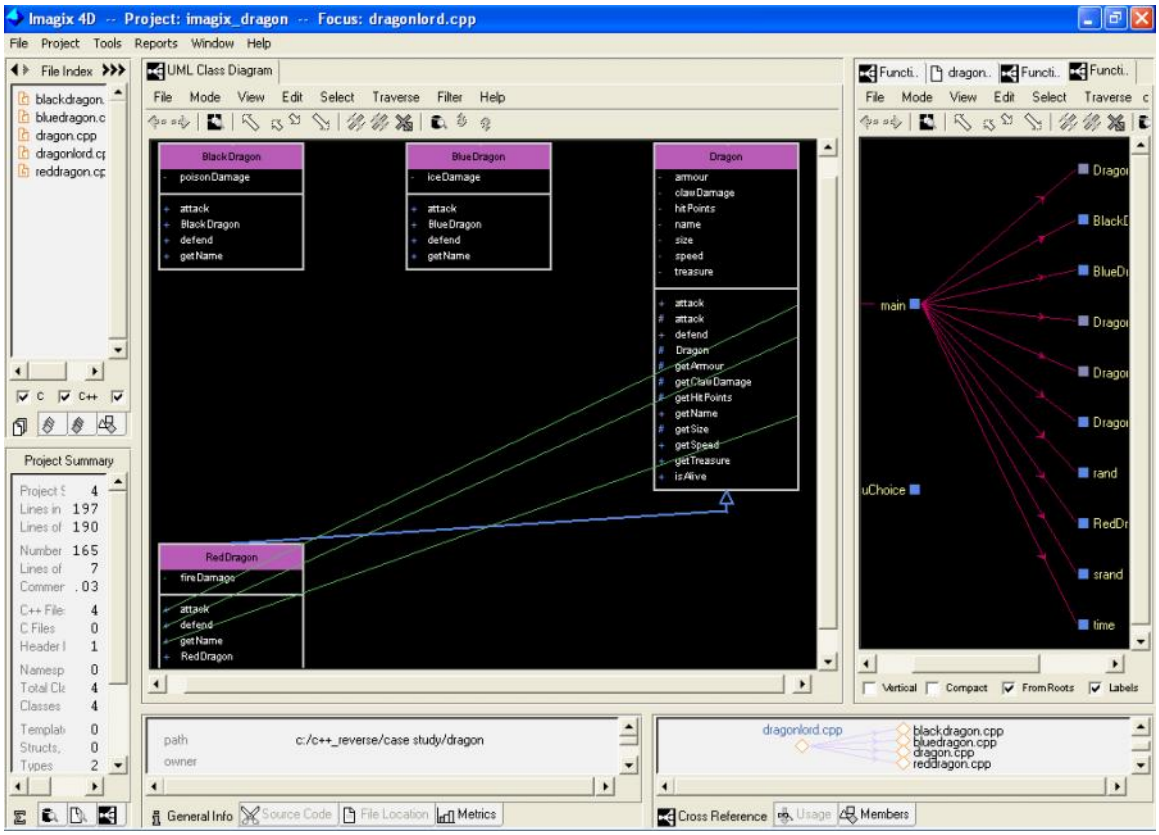

Figure 2: Dragon application in Imagix 4D.

The second project is a large C++ library, namely LibMusicXML, which includes a big set of classes that cover the elements defined by the MusicXML 1.0 dtds (an open XML-based music notion file format). It is an open source project and hosted in the sourceforge website (http://sourceforge.net/projects/libmusicxml/). The library includes 120 files, 196 classes with about 28859 LOC and provided by Visual C++ 6.0.

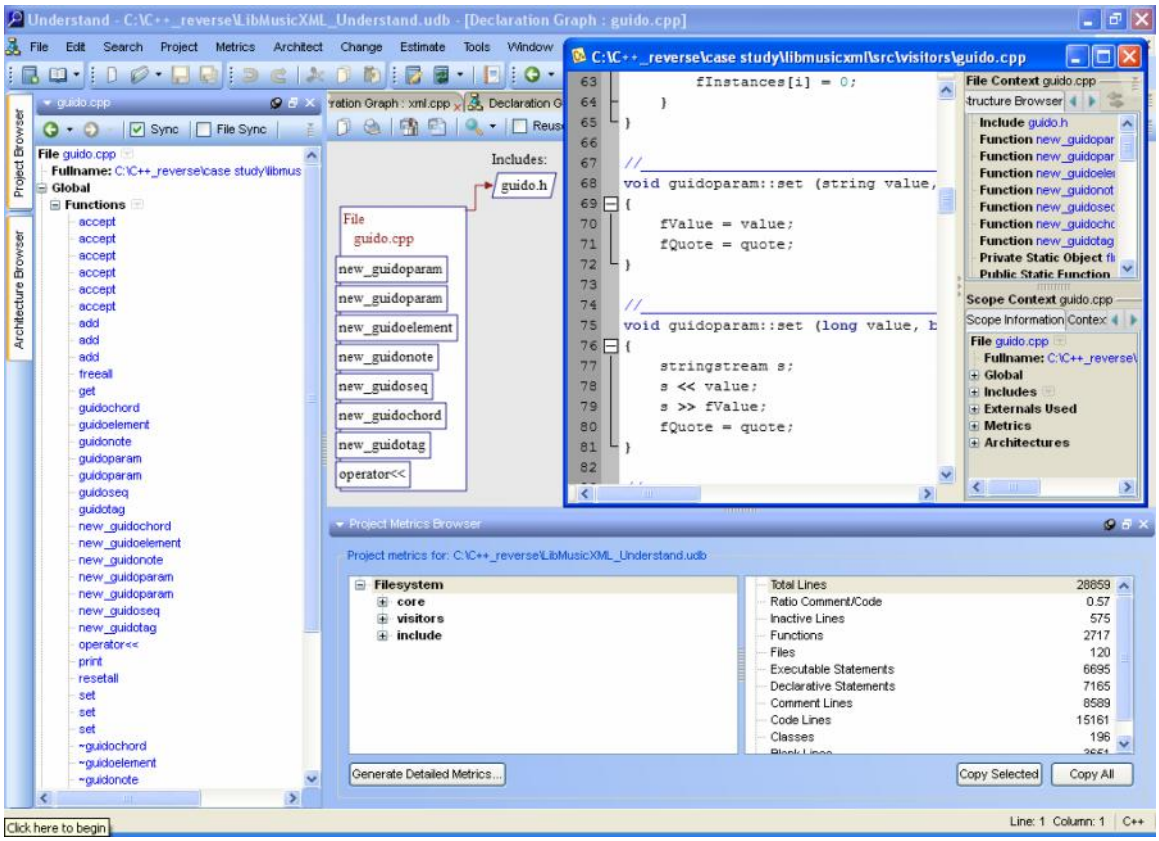

Figure 3: LibMusicXML library in Understand

4.5. Assessment of tools

Each criterion is assessed by one of three methods which I define as follows:

- A list of all possible types of a tool wi thin a particular criterion
- Yes or No to indicate the availability of a feature or capability of each tool
- A four-level scale to evaluate the quality, efficiency of a feature or capability of each tool. They are:
  - "+++": excellent
  - "++": good
  - "+": satisfactory
  - "-": not at all

The table below shows the results of the assessment process:

Assessment criteria	Rigi	Columbus/ CAN	Imagix 4D	Understand
Import/Export				
Importable source code types	C, C++, COBOL	C, C++, MSVC	C, C++, Java, MSVC	C, C++, C#, Java, MSVC, Ada, Pascal, Fortran, assembly, Jovial, PL/M
Project definition types	File	File, Directory	File, Directory, Makefile	File, Directory
Other importable sources	No	Yes	Yes	No
Output formats	No	CPPML, GXL, HTML, RSF(Rigi format), UML XMI, FAMIX XMI	ASCII, RTF, HTML, PNG, VSD (visio files), ps (PostScript)	ASCII, HTML, XML, VSD(Visio files), PNG
Ease of project definition	+	++	++	+++
Analysis				
Incremental parsing	No	No	Yes	Yes
Reparsing	Yes	Yes	Yes	Yes
Fault tolerant parsing	No	Yes	No	Yes
Define and undefine	Project	Project	Project/File	Project/File
Quality of error statements	++	++	++	++
Capability to abort parsing	Yes	No	No	Yes
Parsing results	+	++	++	++
Parsing speed	+	++	++	++
Browsing/ Editing				
Integrated text editor/browser	-	+	+++	++
External editor/browser	Yes	No	Yes	Yes
Control capabilities of text	+	++	+++	+++
The usability of user interface	+	++	+++	+++

Search function	-	-	+++	+++
Highlight capability of source code	No	No	Yes	Yes
Hypertext capabilities	No	No	Yes	No
Representation				
Static or dynamic views	Static views	Static views	Static views	Static views
Layered view	Yes	No	No	No
Filtering, scoping and grouping	++	No	+++	+++
Movement between reports	No	No	No	No
Movement from reports to source code	Yes	No	Yes	Yes
Other capabilities				
Supported platforms	Multiple platforms	Windows	Multiple platforms	Multiple platforms
Integrated metrics	-	+	+++	+++
Change analysis	No	No	No	Yes
Extensibility	++	+++	+	+

Table 1: The result of evaluating the four tools with the chosen criteria

4.6. An analysis of tools

Import/Export

All four tools support examining C, C++ source code. In particular, Understand supports several programming languages from C#, Java to Assembly, Pascal. In addition, thr ee of the four tools (Columbus/CAN, Imagix 4D, Understand) support importing projects written by Visual C++. It is a useful capability because there are a lot of projects made by Visual C++, instead of C++.

Understand provides the easiest way to define a project comparing with other tools. The user only follows steps in the “New project wizard” process to create a new project and import source code by adding files or just a directory. In the case of Imagix 4D and Columbus/CAN you must create a new project first and then use another menu to import source code. Rigi only supports defining a new project by file. All tools support

updating source code into the project. It means that the user can add new source files for his/her current project.

Columbus/CAN provides an excellent export capability which generates the result of the parsing into six different formats. They are efficient for showing both textual (class descriptions) and graphical reports (class diagrams), along with using the result with other tools (Rigi, Famix). However, this tool does not provide capability to store reports hence the user can not view reports in the user interface of the tool. Imagix 4D and Understand also provide good capability in exporting the result of the source code parsing. Textual reports are exported into the formats of ASCII, HTML, RTF, whereas graphical reports are exported into the formats of images (.png) or the Visio tool (.vsd). The limitation of the Rigi tool is that the user can not export the result of the parsing .

Analysis

The parser of Understand supports most of necessary functionalities such as incremental parsing, reparsing, fault tolerant parsing, and ability to abort the parsing, whereas three other tools only support two of the above four functionalities . Rigi does not support incremental parsing and fault tolerant parsing. Columbus/CAN does not support incremental parsing and ability to abort the parsing. Imagix 4D does not support fault tolerant parsing and ability to abort the parsing.

Imagix 4D and Understand support the capability to analyze a single file, along with analyze the whole project. This is very useful because in some cases, the user need to analyze deeply a particular file in order to understand more clearly. In addition, with this capability, the user analyzes a file and continues to find out the files which have relationships with this file. By this way, from an original clue, the user can know the relationships among elements in the system. It is also efficient to analyze the role of a file in the system. Columbus/CAN and Rigi do not support this capability.

The result of the parsing process with Columbus/CAN, Imagix 4D and Understand are good and better than those in Rigi. This conclusion comes from comparing the results of the projects in the case study after using the four tools to analyze them with their architecture and structure which are provided in the project's documents. The Rigi's parser only supports parsing function and data of type "struct" [Bellay and Gall, 1998]. The parser of Columbus/CAN is highly assessed because of the capabilities

to handle templates and to support the precompiled headers technique [Ferenc et al., 2002] which is efficient in reducing compilation time in large projects. The speed of the parsing process with Columubus/CAN, Imagix 4D and Understand in large projects are the same and faster than these in Rigi.

Browsing/Editing

Three of the four tools which are Columbus/CAN, Imagix 4D and Understand, provide an integrated text editor/browser. The editor of U nderstand is the most efficient one. It looks like the code browser of the IDE (Integrated Development Environment).

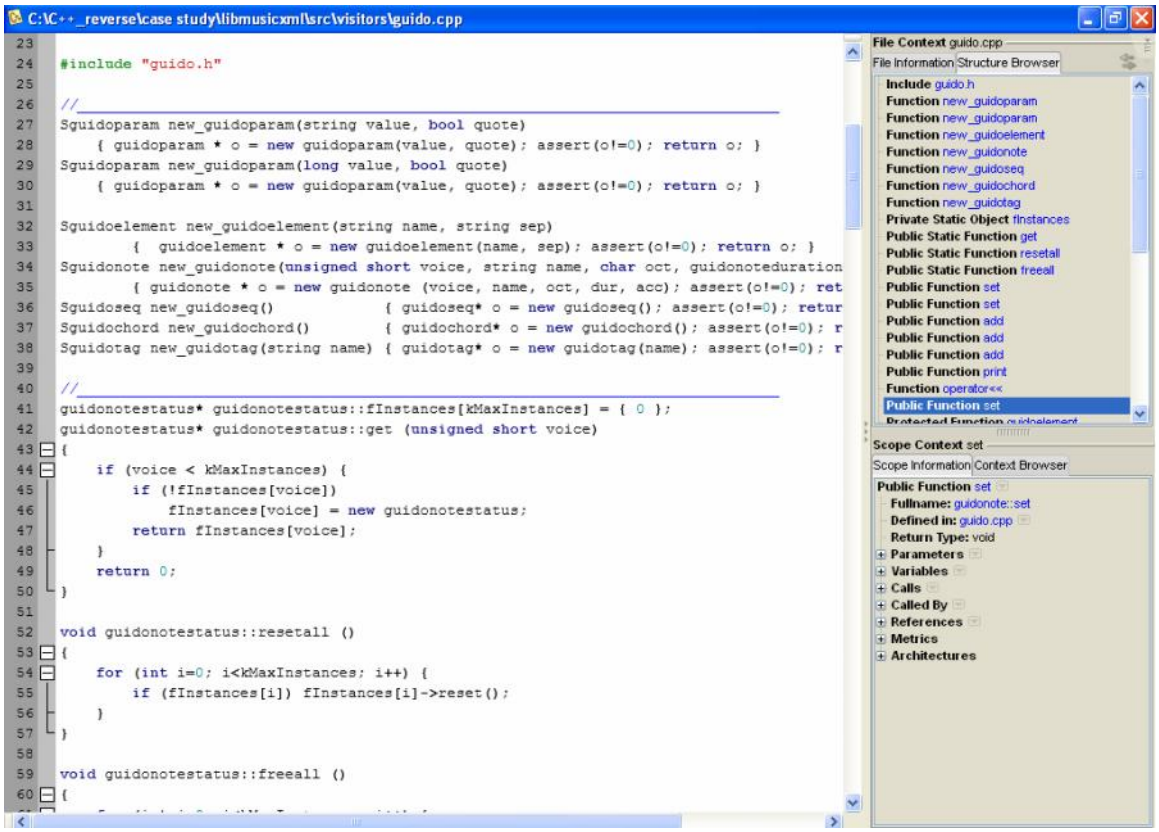

Figure 4: The editor/browser of Understand

In addition to basic capability such as code highlight, line number, text copying and pasting, it provides capability to jump to a particular function, method or line in a source file, and to search correctly any word and then to replace by new word. Moreover, there are other views for the user can see the structure of a fi le, information about scope, context of the file. The only limitation of this editor is that it does not support hypertext capabilities. The editor of Imagix 4D supports this capability. It is very useful for the user

can jump among words in a source file in order to find out clearly the relationships among elements in the file, as well as the structure of the file.

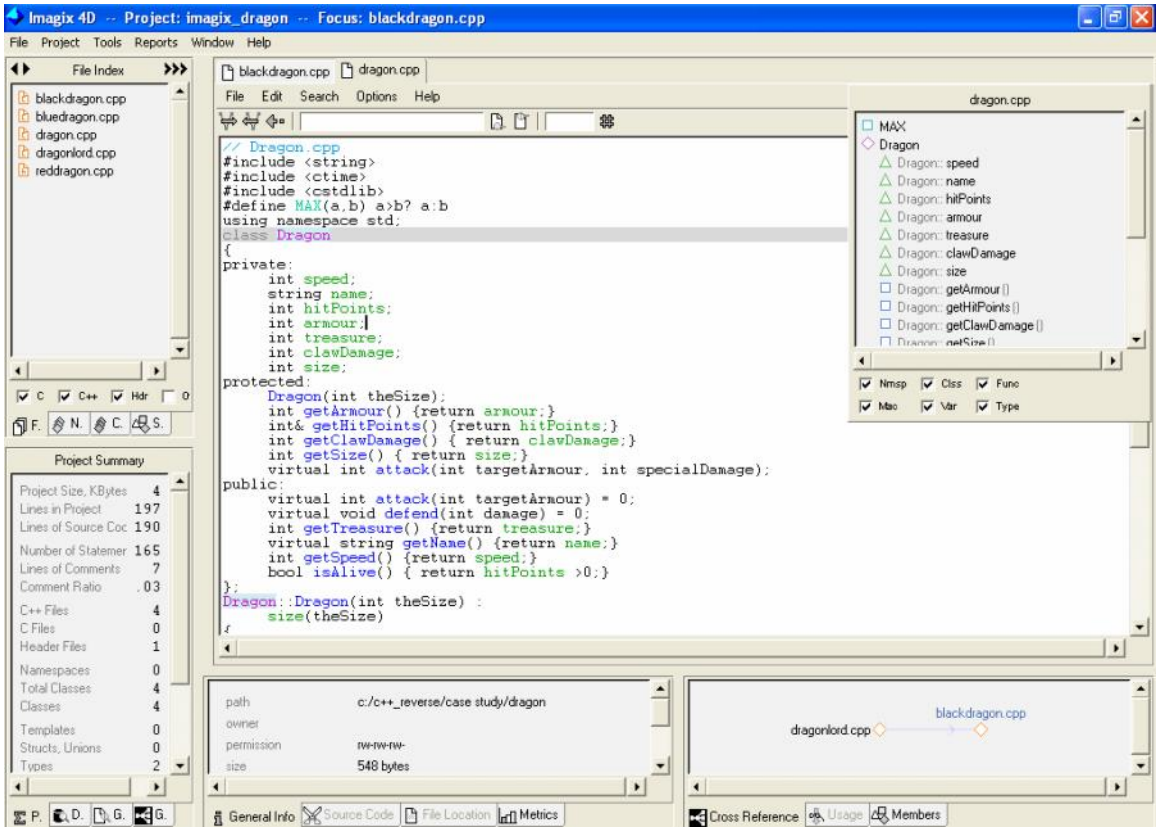

Figure 5: The editor of Imagix 4D

The capabilities of this editor is the same as those of Understand, except it doe s not show line numbers in the first left column of the editor and does not provide more information about the file.

The browser of Columbus/CAN provides only one capability for the user to view a source file. It does not provide any other capabilities su ch as code editing, code highlight, search functionality.

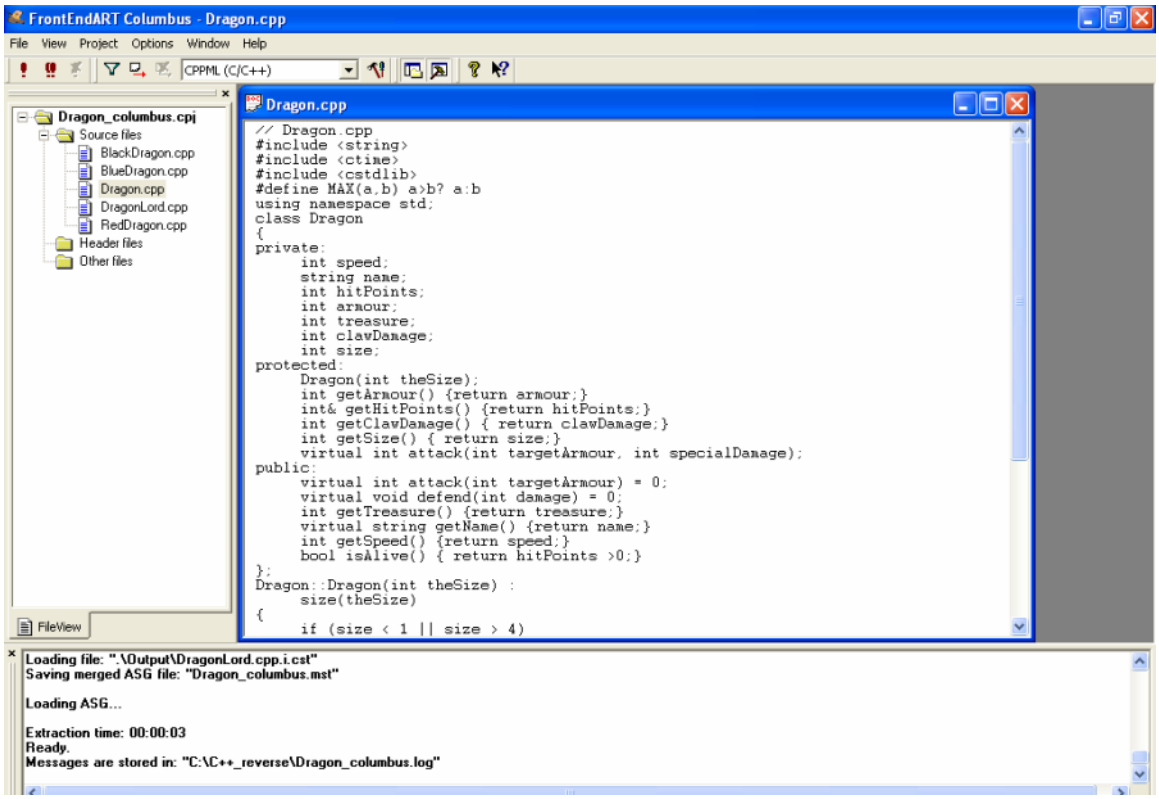

Figure 6: The browser of Columbus/CAN

Representation

In this section, I first provide general view about the representation of the four tool s and then analyze each tool based on the above criteria.

Rigi provides many capabilities in its graph editor. The remarkable characteristic of this tool is viewing capability in many layers with two approaches: m ultiple windows and SHriMP (Simple Hierarchical Multi - Perspective) views [Storey et al., 1997]. In addition, the user can add, edit, delete, modify, and move nodes, arcs in the hierarchy tree. They also collapse or expand a subsystem. It also provides efficient zooming, filtering, scaling, and fitting capabilities which are ne cessary to handle the large and complex structure of the system in graphical reports.

Imagix 4D provides several views about the system in its user interface. For example, the user can choose to view call graph, file diagrams, class diagrams, and control flow. It is very efficient for the user can view and handle directly many types of graphical views at high level of abstraction in the user interface of the tool. Especially, it provides the capability to analyze a single file and then show many types of its

relationships in graphical reports. For example, external functions calling them or external functions they call. We can see more detail for the figure below.

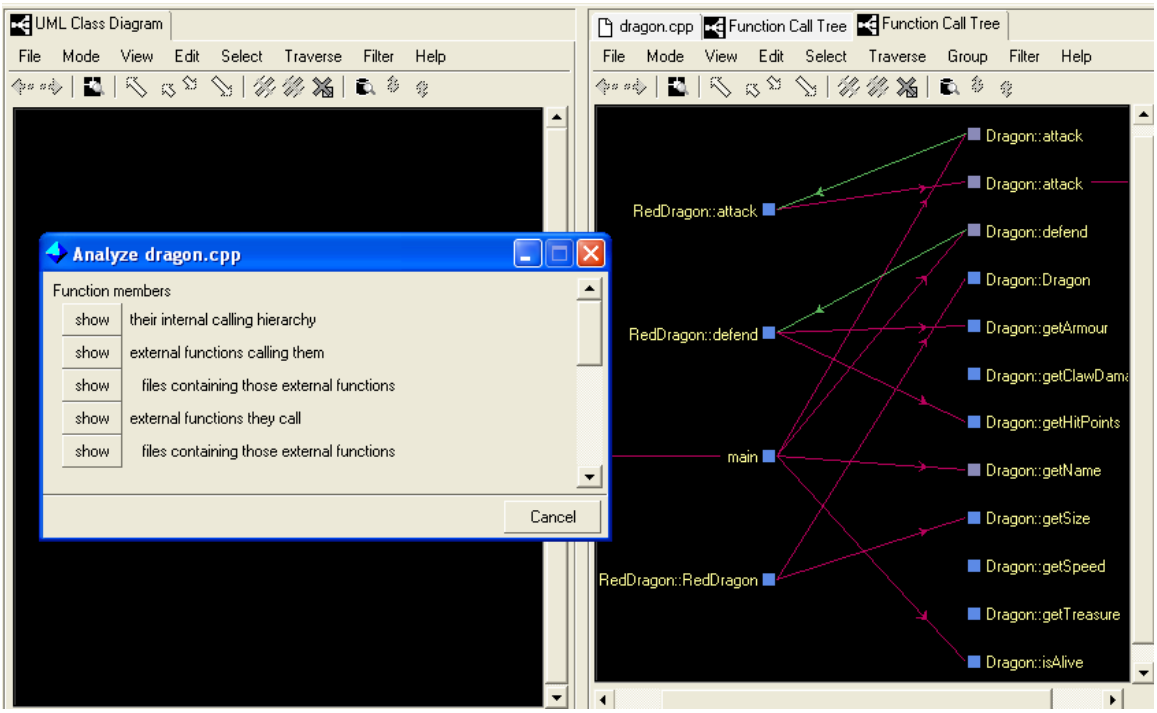

Figure 7: Analyze a file in Imagix 4D

Understand also supports viewing the results of the parsing process in the user interface of the tool. It also provides ability to analyze a file and show the results in graphical reports which are easier for the user to understand. Moreover, the user can handle directly in graphical reports to generate classes, files, methods which have relationships with the file.

Columbus/CAN does not support viewing the results of the parsing process in the tool. Instead of this, it exports the results into six types of format such as HTML, XMI. The figure below shows the content of a file in the HTML format. It only provides information about the attributes, methods, and relationships of a class in the file. The user also see graph, diagrams in other formats such as GXL, XMI. This capability is very useful for generating documents but there are some limitations as the user cannot view representations directly in the tool. For example, the user can not modify the graph, diagram, or use filtering capability to view parts of the whole structure whil e the structure of the system is large and complex, or the user can not link from a point in the graph or diagram to a corresponding point in the source code.

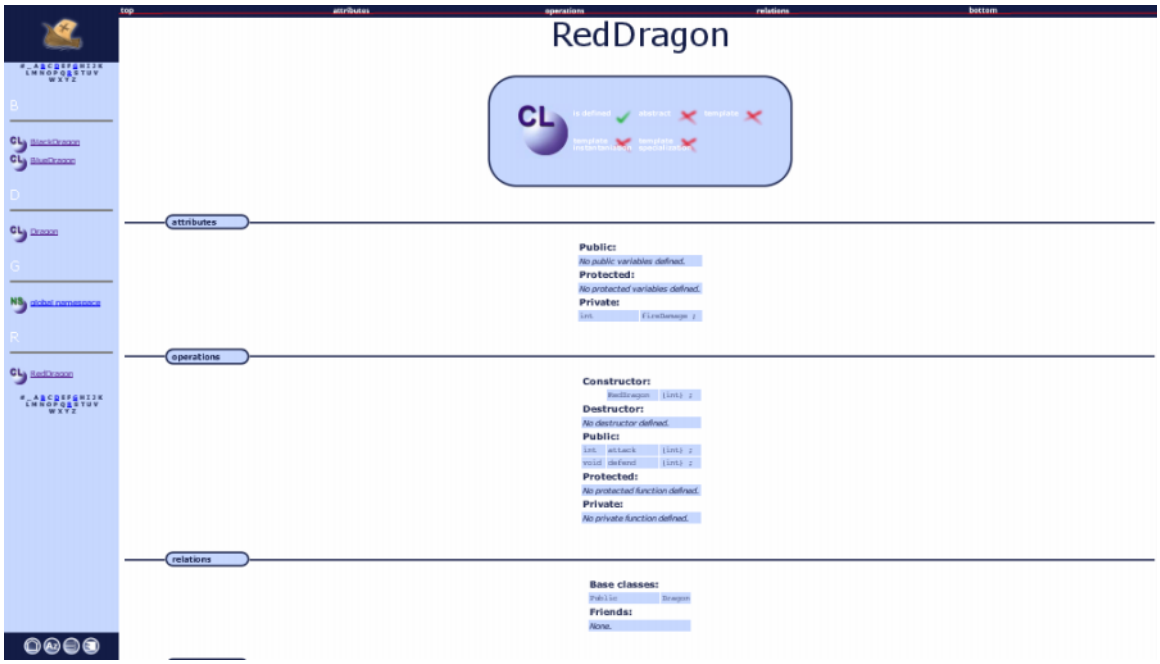

Figure 8: The result of the parsing process with Columbus/CAN in HTM L format

There is no tool which supports the dynamic view. It means that when there are some changes in the source code, the reports cannot reflect these changes. In order to update the report, the user must reparse. This is the limitation of all four too ls although it is not easy to implement this feature.

Rigi is the only tool which provides the ability to view the results in many layers with two techniques: multiple windows and SHriMP (Simple Hierarchical Multi - Perspective) views [Storey et al., 1997]. The second technique is very efficient for viewing the structure of large projects. T he user can see the whole structure of the software system in a nested graph. The algorithm used in this approach is a fisheye view. The fish eye view means that you can see simultaneously the local detail and global context of a graph.

Rigi provides basic capabilities for filtering objects in the graph reports. The user can hide or show the names of nodes in order to reduce the visual clutter when there are a lot of nodes. The user also hides or shows selected groups of nodes, nodes by type or arcs by type. However, rigi is the efficient tool for viewing large graphs because the user can use zoom in or zoom out, especially the user can scale the nodes to fit in a window. The filtering capability of Imagix 4D is also acceptable as it provides ability to hide, isolate

selected objects in the call graph. The user also find objects in the call graph by their attributes (type of file or program elements). The filtering capabil ity of Understand is also the same as two above tools as the user can hide nodes, sub -nodes or collapse sub-nodes. This tool also has zooming capability.

All four tools do not support movement between reports but three of them, except Columbus/CAN, support movement from the reports to source code. The user just click one item in the reports, the corresponding item in the source code will be pointed or highlighted. This capability is very useful for clarifying the structure of parts in source code.

Other capabilities

The ability to run on various platforms makes a tool can be used more widely. Most of the four tools can run on several popular platforms such as Windows, Solaris, Linux, except Columbus/CAN which runs only on Windows.

Both Imagix 4D and Understand provide a lot of software metrics which are very useful in tracking the quality, complexity, and difficulty of software systems. The metrics measures various aspects of software systems in many levels: project, file, function or program unit, class and variable. In addition, the two tools also provide metrics which measure the complexity and difficulty (McCabe cyclomatic complexity and Hastead program difficulty), especially, there are also metrics which measures the class coupling and class cohesion of object-oriented software systems [Understand User guide, 2008]. Columbus/CAN also provides many software metrics at three levels: system, class, and function with the focus on metrics about measuring the class coupling, cohesion and inheritance. Metrics are exports in a file (.csv) and the user can browse by another tool such as Excel. It means that the user cannot see metrics from the user interface of the tool. Rigi provides software metrics for measuring the class coupling and cohesion.

Understand is the only tool from the four tools which supports change analysis capability [Understand webpage]. This capability is very efficient for the purpose of maintaining software systems. The user can compare between two files, two folders in order to know which file or folder is changed from the previous version. In addition, the user can compare between two sections in a file in order to find, for example, why one section of source code run well but another section of source code does not run well.

Columbus/CAN and Rigi are easy to extend features and capabilities, whereas Understand and Imagix 4D are not. The architecture of Columbus is based on plug -ins, hence it is easy to extend core functionalities [Beszédes et al., 1999]. The user can use an easy-to-use plug-in API to write and add new functionalities into the Columbus system or to connect the system with other tools. The architecture of Rigi is based on the architecture namely Programmable Hyper Structure Editor (PHSE) which makes the tool easy to extend core functionalities [Rigi webpage, 2008]. About the capability to link with other tools, Columbus is the only tool which generates reports in the formats of other tools such as Rigi and FAMIX which help the user links more than one tool for his/her work [Ferenc et al., 2002].

4.7. Discussions

In this section, the evaluation results are discussed by comparing with the previous studies. In general, the evaluation result of Rigi is the same as the evaluation results in previous studies. The main advantage of this tool is the capability to represent the structure of a software system in graphical views. Therefore, it is also considered as a visualization tool. Because of this, Columbus/CAN provides an ability to export the result of the parsing process to the format of Rigi (RSF). As a result, the user can use Columbus/CAN, which provides an efficient parser, to parse the source code of the software system and then use Rigi to visualize this result. One main difference between my work and previous studies is that I concentrate much on evaluating the capabilities of Rigiedit that is a graph editor. It is where the user can view, edit, and analyze the structure of the system in a hierarchical tree. Previous studies did not provide results in evaluating the capabilities of Understand hence the evaluation result of this tool in this thesis is new and does not relate to previous studies.

Regarding the evaluation result of Columbus/CAN, in the previous studies, researchers evaluated the previous version of this tool. At that time, it is called Columbus/TDE. They evaluated the capabilities of Columbus tool in the TDE environment which provide the capability to visualize software systems. The remarkable capability of this tool is an efficient parser hence previous studies focused much on this capability. The evaluation result of this capability in this thesis is also the same as in

previous studies. However, I also concentrate on other features and capabilities of this tools which are not evaluated much in previous studies such as import capability, software metrics, export capability and usability.

Regarding the evaluation result of Imagix 4D, because the version of this tool used in previous studies is quite old comparing with the version used in my evaluation work. Therefore, the evaluation result indicates that this tool has been updated with new features and capabilities. The capabilities of import/export, analysis, editing/browsing and representation are also better than those of the old version. For instance, the editor provide more capabilities and all of them work well. There are more options for the user to analyze the structure of the software system in graphical views.

To conclude, the evaluation result in this thesis is more comprehensive and complete than the results in the previous studies. In general, two results are the same. Moreover, all four tools are not only evaluated but also compared hence the evaluation result brings an opportunity to understand deeply and precisely about the four reverse engineering tools and the strengths and weaknesses of each tool as well.

4.8. Conclusions

Each tool provides various features and capabilities and then has different strengths and limitations. After evaluating the four tools, I suppose that there is no single tool which is the best one in all cases. In this conclusion, I summarize and highlight the main features and capabilities of each tool and its strengths and weaknesses as well.

Rigi

Rigi is a free tool, released in the forms of research prototypes, provides basis features and capabilities for the purpose of reverse engineering. Two remarkable capabilities of this tool are (i) techniques in representation such as layered views, SHriMP view which are very useful in large and complex software systems and (ii) extensibility which makes it easy to extend with new features or integrate with other tools. However, it has several limitations in usability, ease to use and efficiency. The user interface is so poor and difficult to use with no integrated source code editor/browser. The parser only supports parsing functions and struct data types, hence it only generates the structure of the

software system in functional views (call graph) and cannot generate s in other views such as class diagrams and control flows. It also pro vides limited numbers of software metrics.

Columbus/CAN

Columbus/CAN is a commercial tool but it also provides a free version for the purpose of academic studies. The strengths of this tool are efficient parsing, export capability and extensibility. The CAN parser in this tool is highly graded because of the capabilities to handle templates and to support the precompiled headers technique [Ferenc et al., 2002] which is efficient in reducing compilation time in large projects. Addtionally, the speed of the parsing process is fast. In the case of export capability, it provides an excellent export capability which generates the result of the parsing process into six different formats. They are efficient for showing both textual reports (class descriptions) and graphical reports (class diagrams), along with usi ng the result with other tools such as Rigi and Famix. In the case of extensibility, the architecture of Columbus is based on plug-ins, hence it is easy to extend core functionalities. The user can use an e asy-to-use plug-in API to write and add new functionalities into the Columbus system or to connect the system with other tools. However, it also has some weaknesses. For example, the user cannot view the results in the tool, hence they cannot analyze them in higher levels of abstractions with graphical reports, or it only runs on Windows platform, or the user interface only provides basis features which do not satisfy the need of the user.

Imagix 4D

Imagix 4D is a commercial tool with many outstanding featu res and capabilities such as many views in representation which displays class diagrams, control flows, flow charts and file diagrams in various windows; analysis capabilities in graphical reports and movement between source code and these reports; quality track with a lot of software metrics; an excellent source code editor with all necessary features such as hypertext, source code highlight, search capability, text control capability, and movement capability to a specific point in the file; exporting the results to various formats; supporting multiple platforms and importing Visual ++ projects.

Understand

Understand is also a commercial tool with an excellent user interface which looks like the IDE of Visual studio. It is customizable, usable, easy to us e, and efficient. The user can

create his/her workplace by organizing the position of windows. Other remarkable capabilities of this tool are reversing combined programming languages and analyzing change compact. It is also very useful in handling large pr ojects. The parser generates correct results with high speed. The user can analyze a file and then represent it in graphical reports. It also provide a lot of software metrics at various levels of a software system such as project, file, class, method, and variable and other metrics in measuring the complexity and difficulty of the system. The main limitation of this tool is that it does not generate the whole structure of the system in graphical reports such as class diagrams or hierarchy trees.

5. Discussions

In this chapter, the reflection of the four reverse engineering tools capabilities on the basic of reverse engineering is discussed and then the strengths and weaknesses of all four tools are mentioned, as well as some suggestions for designing an efficient reverse engineering tool.

5.1. The reflection of the four tools capabilities on the basic of reverse engineering

As defined in the second chapter, reverse engineering is a process of examining a software system to identify its components and their interrelationships and representing it at higher levels of abstraction. In general, all four tools provide support for these tasks such as (i) parsing the source code of the software system to identify its structure and (ii) representing the software system at higher levels of abstraction such as hierarchical graphs, class diagrams, control flows and flow charts. They also provide other capabilities such as software metrics, change analysis, and quality check to support software engineers' tasks. However, the capabilities and features of the four tools have limitations hence they satisfy partly the needs of software engineers. These will be discussed more detailed in the next paragraphs.

Regarding the sub-areas of reverse engineering, the four reverse engineering tools are useful for the tasks of re-documentation of a software system. By using these tools, software engineers are easy to represent the structure of the software system at higher levels of abstraction, even analyze the software system at these levels of abstraction. However, all four tools provide the insufficient and inefficient capability to recover the architecture of the software system. For instance, Understand and Columbus/CAN cannot generate the whole architecture whereas Rigi and Imagix 4D generate inefficiently the architecture of large and complex systems.

Regarding the objectives of reverse engineering, the capabilities of the four tools contribute on the achievement of objectives of reverse engineering. They provide support to (i) coping with the complexity by parsing automatically the large amount of source code, (ii) generating alternative views such as hierarchical graphs, call graphs and class diagrams, (iii) recovering lost information by exporting output to various formats such as HTML, XML, and XMI, (iv) detecting side effects, (v) synthesizing high abstractions and

(vi) facilitating reuse by finding out possible reuse components. However, the capabilities still have limitations. For instance, the four tools are not efficient with large and complex software systems as they cannot generate the accurate architecture of such systems. In addition, they also do not provide techniques to represent the architecture in an efficient way. Regarding the alternative views, they provide a limited numbers of graphical views which are not enough for the software engineer to understand the system.

Regarding the generic reverse engineering process, the four tools are essential for tasks in every phase of the process but their contribution is not much. In the first phase namely "data gathering", these tools are used to examine statically a software system. However, they do not provide the ability to examine dynamically the system. It means the ability to analyze the executing system. In addition, these tools support importing only the source code not other type of data such as documents and experts' knowledge. Therefore, they do not provide the ability to gather sufficiently information of the system. Because of this, they do not provide ability to manage a large amount of knowledge. The second phase plays an important role in a generic reverse engineering process because the comprehensive and consistent management of the large amount of knowledge about the system leads to the success of a reverse engineering project. However, these tools totally do not provide support for this phase. In the last phase, namely "information exploration", these tools are useful when they provide capabilities to represent, navigate and analyze the structure of the system at various higher levels of abstraction. However, these capabilities are applied for static information.

Regarding the reverse engineering methods and techniques, the four tools do not provide an efficient support to them. First of all, these tools provide a limited number of reverse engineering techniques and methods. For example, Imagix 4D and Rigi provide four techniques, Understand provides three ones and Columbus/CAN provides only two ones. You can see detailed information in the table below. Some techniques and methods which are very useful for understanding and maintaining software systems are not provided by the tools such as concept/feature location, clone detection and impact analysis. Secondly, the support capabilities of the four tools are also very limited. They only provide basic capabilities for the user to use the above techniques and methods. For example, using code visualization and design recovery in the four tools are not efficient

when they do not provide an efficient view for the whole architecture of a system. The tools provide only a few options for the user. Finally, the tools only support these techniques and methods in static analysis, not dynamic analysis. The user cannot use dynamic program slicing, dynamic dependency analysis, and dynamic clustering. This leads to the inefficiency of the four tools when using them with real-time, embedded, and client-server applications. In general, Rigi supports code visualization better than other tools. Columbus/CAN only supports dependency analysis and clustering but has several limited. Imagix 4D is remarkable with dependency analysis and clustering. Understand is efficient when using the dependency analysis technique.

	Rigi	Columbus/CAN	Imagix 4D	Understand
Code visualization	Yes	No	Yes	Yes
Program slicing	Yes	No	No	No
Concept/feature location	No	No	No	No
Design recovery	Yes	No	Yes	No
Dependency analysis	Yes	Yes	Yes	Yes
Clustering	No	Yes	Yes	Yes
Clone detection	No	No	No	No
Impact analysis	No	No	No	No

Table 2: Reverse engineering methods and techniques

In summary, software developers should use the four tools to analyze the source code of a system and then identify its components and their interrelationships. They also use them for other tasks such as tracking the quality of the system with software metrics, and analyzing change impact. Software designers should use them to export the structure of the system to various formats such as HTML, XML, and XMI. However, the four tools only provide the capability to analyze statically the system hence they cannot satisfy the software engineers' needs. Especially, these tools do not provide an integrated environment for software engineers to do all their tasks on it. They are only tools, not frameworks which support the whole generic reverse engineering process. In addition, software engineers cannot write lines of code and then debug and run on these tools. It means they support only analyzing code, not synthesizing code.

5.2. Strengths of the four reverse engineering tools

5.2.1. Representation of software at higher levels of abstraction

All four tools support the representation of software at higher levels of abstraction. In particular, Rigi represents the structure of software in a hierarchy tree with nodes and arcs [Rigi User's manual, 1998]. Nodes represent for artifacts in software and arcs represent for their relationships. Columbus/CAN exports the output to the format of UML diagrams (.xmi) and we can view it by any tools which support handling UML diagrams. Imagix 4D represents the structure of software at several higher levels of abstractions such as call graphs, control flows, file diagrams, class diagrams and flow charts as in the figure below. Understand represents the structure of software in class diagrams, flow charts.

It is very difficult for the software developers to understand the structure of software when examining manually source code because of the large amount of source code and the complexity of software. The structure of the software system will be easier to understand by the software engineer if it can be represented at higher levels of abstraction. For example, with class diagrams, the software engineer is easy to find out all classes in an object-oriented software system and especially, their interrelationships which build the structure of this software system. Consequently, this capability of all four tools is essential for understanding the structure of software. It is also a main nation of reverse engineering tools.

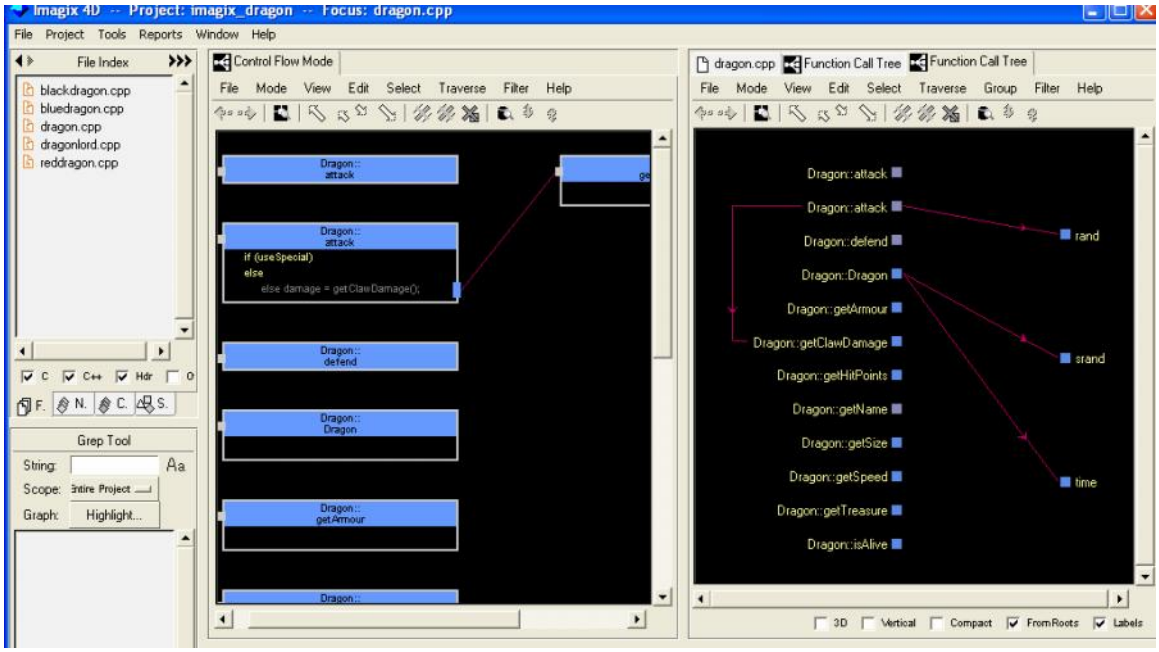

Figure 9: Representation of software at higher levels of abstraction in Imagix 4D

5.2.2. Analysis of software at higher levels of abstraction

Reverse engineering tools provide not only the ability to represent software at higher levels of abstraction but also the ability to analyze software at these levels of abstraction. The software engineer is able to work in graphical reports which display the results of parsing process such as modifying items or generating new items which are relevant with the root element. All chosen tools except Columbus/CAN provide the analysis capability of software at higher levels of abstraction. In particular, by using Rigi, the user is able to modify nodes or arcs of the hierarchy tree, which represents the structure of software, in a graphical editor, namely "rigiedit" [Bellay and Gall, 1998]. The user also collapse subsystems of the whole system. This capability is efficient when handling large and complex hierarchy trees. By viewing subsystems of such these systems, the user is easy to understand the structure of the software. In the case of Imagix 4D, it provides many capabilities in handling in graphical reports. For example, in a class diagram, the user can generate all relationships of a file such as external functions it calls, external functions calling it, it internal call hierarchy (see the figure below). Like two above tools, Understand provides ability to analyze a file or a class and then generate its relationships. This is very useful for the user to understand the role of a file or a class in the whole

system. It is also very useful when finding out the structure of the whole system from an original clue which is a file or a class or in handling large and complex systems.

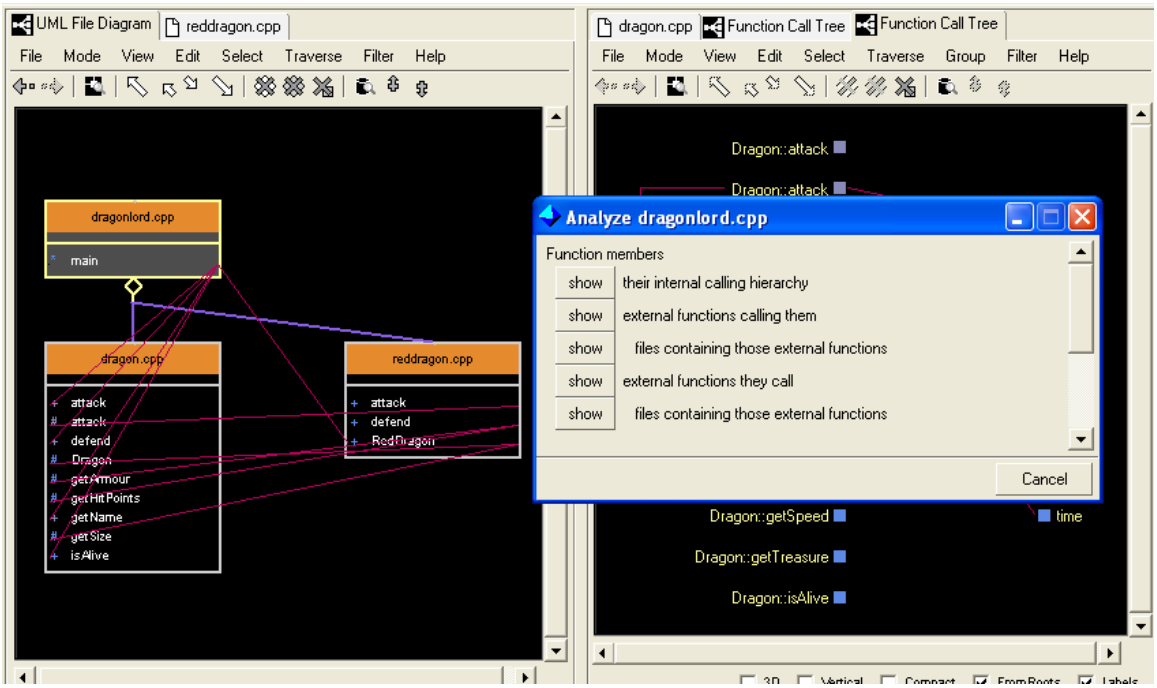

Figure 10: Generate all functions, files which are relevant to a file in Imagix 4D.

5.2.3. Documentation generation

Documentation is one of the most important information and knowledge about a software system which helps the software engineers understand the software system. Therefore, creating documentation in software projects is a compulsory task and often takes much effort and time. However, after a long time being used, there are often some changes in the system which makes it different from original version. Documentation is not up-to-date and it does not include explicit knowledge about the system. Hence, the ability to generate documentation from the source code is one of the remarkable features of reverse engineering tools. Additionally, understanding the structure of the system from its documentation is easier than from its source code because documentation includes knowledge about the system at higher levels of abstraction such as diagrams, charts which are easier to understand than source code which is understandable by software developers. Three of the four tools (Columbus/CAN, Imagix 4D, Understand) provide the documentation generation capability. Columbus/CAN is the tool which generates documents in six types of format such as CPPML, GXL, HTML, RSF (Rigi format),

UML XMI, and FAMIX XMI [Ferenc et al., 2002]. As a result, these documents are able to store and used later by another tool. It also generate documents in the format of other tools such as Rigi and FAMIX, hence the user can use these tools to analyze the system. Moreover, it generates documents in both textual and graphical reports. Imagix 4D and Understand generate documents in the ASCII, HTML, and XML formats [Understand User guide, 2008]. The user can also save diagrams, charts in the formats of image such as PNG, BMP and in the format of the Visio tool, a famous tool from the Microsoft software company. They also provide ability to convert them into printed versions and the user can choose to print them directly from the tool.

5.2.4. Software metrics

All four tools provide software metrics which are necessary in tracking the quality, complexity and difficulty of software systems. However, the numbers of metrics in each tool are different. Imagix 4D and Understand provide a lot of software metrics to measure various aspects of software systems in many levels: project, file, function or program unit, class and variable. In addition, the two tools also provide metrics which measure the complexity and difficulty of the system (McCabe cyclomatic complexity and Hastead program difficulty), especially, there are also metrics which measures the class coupling and class cohesion of object-oriented software systems [Understand User guide, 2008].

Columbus/CAN also provides many software metrics at three levels: system, class, and function with the focus on metrics about measuring the class coupling, cohesion and inheritance. Metrics are exported in a file (.csv) and the user can browse by another tool such as Excel [Ferenc et al., 2002]. It means that the user cannot see metrics from the user interface of the tool. Rigi only provides software metrics for measuring the class coupling and cohesion but these metrics are very efficient in measuring object-oriented software systems [Rigi webpage, 2008].

5.2.5. Change analysis

Understand is the only tool from the four tools which supports change analysis capability [Understand webpage, 2008]. This capability is very efficient for the purpose of maintaining software systems. The user can compare between two files, two folders in order to know which file or folder is changed from the previous version. In addition, the

user can compare between two sections in a file in order to find, for example, why one section of source code run well but another section of source code does not.

5.2.6. Quality checks

Quality checks help the user to identify potential problems which occur in the run-time execution of their software. It provides capabilities to analyze data flow of the source code in order to find out problems of data access, concurrency control. The user can also review possible conflicts in real-time, embedded, and multi-threaded systems. In fact, Imagix 4D is the only tool which provides this capability [Imagix 4D webpage, 2008].

5.3. Limitations of the four reverse engineering tools

5.3.1. Inefficiency of overall architecture of software

Columbus/CAN, Understand cannot generate the overall architecture of a software system from its source code. The tools provide ability to generate high levels of abstraction of each component, file, or class in graphical reports. The user can analyze in these reports to find out all relationships of a component in the system but he/she cannot have a comprehensive view about the structure of the whole system. Rigi uses a hierarchy tree to represent the structure of the system with new techniques such as layered views and SHriMP views [Storey et al., 1997]. However, it is not efficient in large projects and it needs much more investigation and research to make it efficient in the terms of usability, effectiveness and ease of understand. Imagix 4D also has many weaknesses in generating the whole structure of software. The result is not correct in some cases and it lacks many capabilities in handling this case such as filtering, grouping, scoping, zooming, and layered view.

5.3.2. Insufficiency of graphical views

All four tools can generate several graphical views such as call graphs, flow charts, control flows, class diagrams and file diagrams but these views are not sufficient enough to have a comprehensive view about a software system. Additionally, information in graphical views is copied from information in the source code. The tools do not generate additional information in these views.

5.3.3. Non-Integration with the IDEs

IDEs (Integrated Development Environments) are frequently used by software developers for their daily works in developing software. Therefore, one of the efficient ways for reverse engineering tools to be widely used is to integrate reverse engineering tools into the IDEs. However, all four reverse engineering tools are unable to do this. They can only provide capabilities to analyze source code, but not to synthesize, debug and build source code. Software developers must use one of the IDEs for software projects and they are really familiar with them. Therefore, they often do not want to use other tools such as reverse engineering tools while they do not have necessary skills for using them and they do not want to pay much more money for them.

Since software developers are familiar with the user interface of IDEs, reverse engineering tools will be easy for them to use if their user interfaces are look like those of IDEs. However, there is only one of the four tools which is Understand with the user interface is the same as the user interface of Visual Studio, the popular IDE from Microsoft.

5.3.4. Inefficiency of graphical views with large projects

All four tools do not support efficiently graphical views with large projects although Rigi provides many techniques in dealing with this problem such as layered views and SHriMP (Simple Hierarchical Multi-Perspective) views. As you could see in the figure below, when there are a lot of items in graphical reports, it is hard for the user can recognize items and their relationships. Filtering and grouping capabilities are not efficient in this case either.

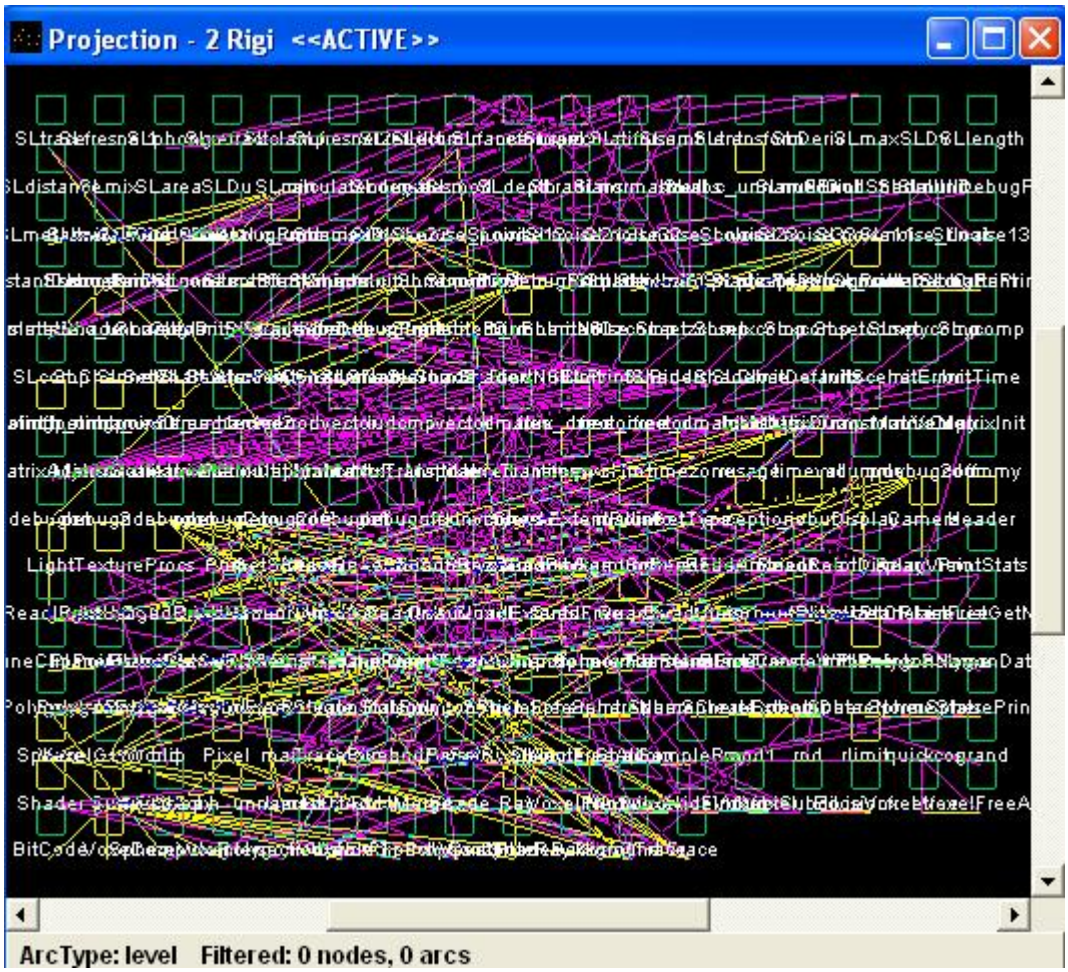

Figure 11: Representation in Rigi with a large and complex software system

5.3.5. Unavailability of dynamic views

All four tools do not support dynamic views. Dynamic views mean that when there are some changes in a view, other views will reflect these changes. This feature is very useful because when the user make some changes in source code, other views such as class diagrams, control flows, call graphs will change automatically. The user does not need to care about the consistency of other views. Since all four tools only support static views, when the user edits something in a source file, he/she must reparse this file in order to reflect these changes in other views. Moreover, the main limitation of this case is that when there are some changes one of the following views: class diagrams, control flows, call graphs, and flow charts, other views cannot reflect these changes. Therefore, the data is not consistent in all views. However, it is very difficult to implement this feature in

reverse engineering tools because managing traceability among other views is hard to obtain.

5.3.6. Unavailability of dynamic analysis

All four tools provide the capability to analyze static information such as source code. They do not provide the capability to analyze executing systems. Analyzing systems when they are running helps us to have knowledge about the interactions between components in the system, types of messages and protocols used and the external resources used by the system [Tilley, 1998]. This is very useful in examining distributed, real-time and client-server applications. Therefore, the four tools are not efficient in such types of application.

5.4. Suggestions for designing an efficient reverse engineering tool

5.4.1. Import/Export

First of all, the import capability of a reverse engineering tool should be usable and easy to use. The user just click on an "import" function in the menu of the tool and a process will be taken automatically with some steps. In each step, a window will be displayed which includes not only boxes, buttons for the user to import source code but also instructions which helps the user know how to do. In additon, it should support importing source code with several options such as adding all files in a directory, adding files having a particular extension, and adding files having a defined starting name or a particular string in the file name. It is necessary because in some cases, software developers only need to analyze some groups of files, not the whole files of a software system. One more thing, after importing source code, the tool still provides the ability to add new files into the project. It should also support importing projects created by frameworks such visual studio, Qt. In summary, the import capability of the reverse engineering should look like the import capability of IDE tools so that the software developer does not feel so confused when using it.

The export capability in reverse engineering tools is very essential due to the purposes of storing and continuous handling the output with other software tools, especially reverse engineering tools. Therefore, a reverse engineering tool should support

exporting the output to various formats: textual formats (description about the results in detail), graphical formats (diagrams, charts, graph trees, images), standard formats (XML, HTML, among other things), formats of most popular CASE tools (UML tools, IDE tools), and formats of other reverse engineering tools.

5.4.2. Analysis

Firstly, the source code parser of a reverse engineering tool should support all effective functionalities such as incremental parsing, reparsing, fault tolerant parsin g, and abortable parsing.

Secondly, it should provide an efficient capability to recognize exactly elements and their interationahips in the source code written by object-oriented programming languages and other exceptions, for instance, templates in C++. New techniques should be investigated to improve the speed of the parsing process which is very important in handling extremely large projects.

Finally, a reverse engineering tool should support the capability to analyze a single file, along with the whole project. This is very useful because in some cases, the user need to analyze deeply a particular file in order to understand it more clearly. In addition, with this capability, the user analyzes a file and continues to find out the files which have relationships with this file. By this way, from an original clue, the user is able to know the relationships among elements in the system. It is also efficient to analyze the role of a file and its impact on the system.

5.4.3. Editing/Browsing

The browser of a reverse engineering tool is not only a place for the user to view source code but also to edit source code. Therefore, it should look like a code editor in IDEs with all necessary capabilities such as (i) hypertext capability, (ii) code highlight, (iii) line number, (iv) text copying and pasting, (v) the capability to jump into a particular function, method or line in a source file, and (vi) search correctly any word and then replace by new words. It also provides ability to link to external text editors.

5.4.4 Representation

All four chosen reverse engineering tools do not support dynamic views which are usefut to improve the traceability among various levels of abstraction. A n efficient tool should

support not only movement between source code and a specific view but also movement among different views. For example, the user can switch from one point in a call graph to a corresponding point in a class diagram, or in control flow. Moreover, whenever there is a new change in a view, other views should be updated. Regarding the user interface, a tool should support multiple views. Each view is a highly customizable window. For instance, the user can generate a call graph, a control flow, a class diagram in various views and can change the position of these views in order to see all views in the user interface. With this capability, the user can see changes in each view when there is a change in another view.

It is very difficult to represent efficiently the whole structure of a large and complex software system. It can include a lot of items, elements in each view. Hence, a reverse engineering tool should provide excellent capabilities in filtering, grouping, scoping and zooming. Using layered views is another way to make complex views more understandable. Applying the "divide and conquer" algorithm is also a solution in this case. The whole structure system will be showed by a tree, a diagram or a chart of sub-systems. The user can click on each sub-system in order to view the structure of this sub-system in a new view.

5.4.5 Other capabilities

First of all, a reverse engineering tool will be used widely if it supports multiple platforms. There is now no platform which is satisfied by all users hence they are using different platforms such as Windows, Solaris, and Linux-based platforms. In addition, a reverse engineering tool should support multiple users in order to be used by more users. This is efficient in the case of a project team or a software company. Finally, the architecture of a reverse engineering is easy to extend new features or to integrate with other popular CASE tools. It should be a component-based architecture with a core platform.

Secondly, software metrics play an important role in tracking software quality. Therefore, a reverse engineering tool should provide a lot of software metrics at various levels of a software system such as project, namespace, file, class, method and variable. Especially, it should provide metrics in measuring the complexity and difficulty (McCabe cyclomatic complexity and Hastead program difficulty) of the software system and the

class coupling and cohesion which is very important in assessing object -oriented software system.

Thirdly, for the purpose of software maintenance, a reverse engineering tool should support change analysis. The user can compare between two files, two folders in order to know which file or folder is changed from the previous version. In addition, the user can compare between two sections in a file in order to find out any changes and differences, for example, why one section of source code run well but another section of source code does not run well.

Fourthly, It should provide better traceability which helps the user not only realize the relationships among files in project, classes in the pr oject, and among other things but also analyze the impact of these relationships. Moreover, a reverse engineering tool should support traceability in different levels of abstraction. In order to do this, a reverse engineering tool should support importing other sources about the software system such as requirement specification, architecture document. A reverse engineering tool will update automatically these documents by analyzing source code and provide traceability among elements of software system at va rious levels of abstraction.

Fifthly, it should provide support to reverse engineering methods and techniques such as program slicing, clone detection, feature/concept location, impact analysis and especially design recovery. The ability to recover the ar chitecture of the system is very useful for understanding the system because the architecture often provide a comprehensive view about the system.

Finally, the support of dynamic analysis methods leads to the efficiency of a reverse engineering tool when a nalyzing distributed, real-time and client-server software systems.

6. Conclusions and Future Work

In this thesis, I presented my work from evaluating the capabilities and features of the four reverse engineering tools for C++ applications: Rigi, Columbus/CAN, Imagix 4D, and Understand. I first presented background knowledge about reverse engineering such as the definition of this term, its sub-areas and objectives and then presented my work in literature review. The latter consisted of various reverse engineering tools reviews and evaluations of their capabilities and features. Bearing in mind these previous reviews, I evaluated the four tools mentioned above by using them to examine two different types of C++ applications: a small code application and an extremely large code library. Moreover, I created evaluation criteria which could evaluate the following support features of the tools outlined here in main categories: *import/export, analysis, browsing/editing, representation and other capabilities* and sub-categories. After evaluating the capabilities of these tools in the above five categories, I derived the strengths and limitations of the four tools. Upon those I drew conclusions and outlined suggestions for designing an efficient reverse engineering tool which would outperform the existing ones.

The four popular reverse engineering tools which I examined in detail in this thesis are very useful for the purposes of software maintenance, re-engineering, re-documentation, and code reuse. They provide designers, programmers and maintainers who are the tools' most frequent users with many software quality capabilities for their work and for documenting their work tasks. These include, for instance, the following: (i) analyzing automatically the source code of a software system and (ii) representing the structure of this system at higher levels of abstraction such as call graphs, flow charts, control flows, and class diagrams. These tools, however, have not been widely used because of limitations and inefficient features and capabilities they have, as mentioned in this thesis but also in other earlier scientific works. When searching articles for the literature review, I found that there have been a few articles which presented the work in evaluating and comparing the capabilities of such tools. The personal motivation, therefore, directed the decision to evaluate and compare the capabilities of the four

widely used reverse engineering tools, which support C++ programming languages in order to answer the three main research questions, also outlined below.

The first question was "What are the features and capabilities of the four reverse engineering tools for C++ applications". I have found that their features and capabilities are different. There is no single tool which could be declared the best in my evaluation. The next paragraphs, though, summarize the evaluation results and provide an answer to the first research question by comparing and contrasting the findings.

Two remarkable capabilities of the Rigi tool are its techniques in representation such as layered views and the SHriMP view which are very useful in the case of large and complex software systems. Extensibility is also of outmost importance because it makes Rigi easy to extend with new features or/and integrate with other tools. In addition, the particular tool also provides: (i) software metrics for measuring class cohesion and coupling, (ii) a graph editor for handling graphical reports, and (iii) ability to customize the user interface.

On the other hand, the remarkable capabilities of the Columbus/CAN tool are (i) efficient parsing, (ii) export capability and (iii) extensibility. The CAN parser in this tool has been highly graded because of its capabilities to handle templates and to support the precompiled headers technique [Ferenc et al., 2002], which is efficient in reducing compilation time in large projects. In the case of export capability, it provides an excellent export capability which can generate the result of the parsing process into six different formats. They are efficient for showing both textual reports (class descriptions) and graphical reports (class diagrams), along with using the result with other tools (Rigi, Famix). In the case of extensibility, the architecture of Columbus is based on plug-ins, hence it is easy to extend core functionalities. The user can utilize an easy-to-use plug-in API to write and add new functionalities into the Columbus system or to connect the system with other tools.

The main features that Imagix 4D provides are outlined next and they consist of significant help in many functions and tasks of the software developer: (i) many views which displays class diagrams, control flows, flow charts and file diagrams in various windows; (ii) analysis capabilities in graphical reports and movement between source code and these reports; (iii) quality track with a lot of software metrics; (iv) an excellent

source code editor with all necessary features such as hypertext, source code highlight, search capability, text control capability, and movement capability to a specific point in the file; (v) exporting the results to various formats; and (vi) supporting multiple platforms and importing Visual ++ projects.

The fourth tool in my selection and evaluation list was Understand. Firstly, this tool provides an excellent user interface, which looks like the IDE of Visual studio. It is customizable, usable, easy to use, and efficient, also according to other reviews and evaluations. Two other remarkable capabilities of this tool are combined programming languages analysis and change analysis. The tool's features are also very useful in handling large projects. The parser generates correct results with high speed. The tool user can analyze a file and then represent it in graphical reports. A lot of software metrics are provided at various levels of a software system such as project, file, class, method, and variable and other metrics in measuring the complexity and difficulty of the system.

The second question was "What are the strengths and limitations of the four reverse engineering tools for C++ applications". Common strengths of these tools are (i) representation of software at higher levels of abstraction such as class diagrams, call graphs, and control flows; (ii) analysis of software at higher levels of abstraction with ability to work in graphical reports in order to generate related items or move to a corresponding item in source code; (iii) documentation generation in many formats such as HTML, XML, and XMI; (iv) tracking software quality with a lot of software metrics at various levels of a software system such as project, file, class, method and variable; (v) change analysis with the ability to compare items in different files, files in various folders; and last but not least (vi) quality checks with ability to identify potential problems which occur in the run-time execution of their software. In contrast, common limitations of these tools are (i) inefficiency of overall architecture of software; (ii) insufficiency of graphical reports with only class diagrams, control flows, flow charts and call graphs; (iii) non-integration with IDEs; (iv) inefficiency of graphical views with large projects; (v) unavailability of dynamic views and (v) unavailability of dynamic analysis.

The last question was "What should be the features and capabilities of an efficient reverse engineering tool for C++ applications". These, according what features exist and

according to what desirable features and capabilities could cover current needs, could be described in the following five main categories: import/export, analysis, editing/browsing, representation and other capabilities. In the case of the import/export feature, for instance, the import capability of a reverse engineering tool should be usable and easy to use. The user only needs to click on an "import" function in the menu and a process will be taken automatically with some steps. It should also support importing projects created by frameworks such visual studio and Qt. Moreover, a reverse engineering tool should support a function that could export the output into various formats: textual formats (description about the results in detail), graphical formats (diagrams, charts, graph trees, images), standard formats (XML, HTML, among other things), formats of most popular CASE tools (UML tools, IDE tools), and formats of other reverse engineering tools. In the case of the analysis capability, a main reverse engineering function, the source code parser of a reverse engineering tool should support functionality with all effective capabilities such as incremental parsing, reparsing, fault tolerant parsing, abortable parsing. An efficient reverse engineering tool should also be able to parse the structure of the source code written by object-oriented and other programming languages, and other exceptions such as templates in C++. Moreover, novel techniques should be investigated to improve the speed of the parsing process which is very important in handling extremely large software projects. In addition, a reverse engineering tool should support the capability to analyze a single file, along with to analyze the whole software projects. In the case of the editing/browsing feature, it should look like a code editor in IDEs with all necessary capabilities such as hypertext capability, code highlight, line number, text copying and pasting, capability to jump to a particular function, method or line in a source file, and search correctly any words and then replace with new words. It should also provide the ability to link to external editors. In the case of the representation capability, a new tool should support not only movement between source code and a specific view but also movements among different views. Moreover, whenever there is a new change in a view, other views should be updated. It is very difficult to represent efficiently the whole structure of a large and complex software system. It can include a lot of items, elements in each view. Hence, a reverse engineering tool should provide excellent capabilities in filtering, grouping, scoping and zooming. In

the case of other capabilities, a reverse engineering tool will be used widely if it supports multiple platforms. It also should support multiple users in order to make it be used by more users. This is efficient in the case of a project team or a software company. Moreover, the architecture of a reverse engineering should be capable to extend new features or to integrate with other popular CASE tools. It should be a component -based architecture with a core platform. Software metrics play an important role in tracking software quality. Therefore, a reverse engineering tool should provide a lot of software metrics at various levels of a software system such as project, namespace, file, cl ass, method and variable. Especially, it should provide metrics in measuring the complexity and difficulty (McCabe cyclomatic complexity and Ha lstead program difficulty) of the software system and the class coupling and cohesion which are very important in assessing object-oriented software systems. For the purpose of software maintenance, a reverse engineering tool should support change analysis. Moreover, a reverse engineering should support traceability in various levels of abstraction.

A last comment I would like to draw here as rather as an observation is that the country of the tool origin indicates the particular software development culture that the tool is exposed at. The four tools, therefore, addressed very different and very common needs regarding the national software industry they belong to. In the four chosen tools I evaluated, Imagix 4D and Understand are made by software companies in United States whereas Rigi is made by a University in Canada and Columbus/CAN is made by a commercial company in Hungary. In my opinion, Imagix 4D and Understand are better than Columbus/CAN in usability and efficiency. The features and capabilities of Imagix 4D and Understand are also many more than those in Columbus/CAN. Rigi also takes much effort, time to research, implement and test by members at a university in Canada. Therefore, someone could conclude that reverse engineering tools are constructed and investigated much more carefully in United States.

This thesis´ aim and motivation have been to provide a valuable, comprehensive and detailed evaluation and comparison of the capabilities of the four popular in use reverse engineering tools. The results of this work can be useful for those who want to find a suitable reverse engineering tool for their software development and maintenance tasks. The thesis also highlighted the strengths and limitations of the four reverse

engineering tools and provided suggestions for designing an efficient reverse engineering tools.

There are many software tools and reverse engineering tools in the market, but there are not many recent evaluations in their strengths and weaknesses. Trying to address this need while proceeding with my thesis work, I encountered some problems. Firstly, evaluating and comparing reverse engineering tools seems to be a wide knowledge topic because it requires the tool evaluator to have enough, at least sufficient knowledge about many fields in software engineering such as reverse engineering, object-oriented programming language, UML, code parsers, CASE tools, software quality, and software maintenance. Secondly, it took a considerable amount of time for me to find out an open source project which released both source code and documents so that I could check the results which are generated by reverse engineering tools with the results in the documents. Last but not least, two of the four chosen tools are commercial ones. Therefore, I only have had temporary licenses in 15 days to use them.

In the future, I expect more updated works in evaluating the capabilities and features of reverse engineering tools. Because of the time limitation in the thesis work, someone cannot evaluate deeply some capabilities such as code parsers, storing capabilities and representation capabilities within large projects. I also hope that these tools would be evaluated with various types of applications. In addition, according to this and other related research works' findings, these tools should receive more attention and investigation, so they could be used widely by improving their capabilities and features. Such quality features are (i) creating much more views in higher levels of abstraction, (ii) providing the ability to import other sources such as requirement specification, architecture, and user interface design and then supporting traceability among various levels of abstraction, (iii) creating a workplace for both software analysis and software synthesis; (iv) supporting combined programming languages, and (v) recovering the architecture of software systems.

References

[Bellay and Gall, 1998] Bellay, B., Gall, H., An evaluation of reverse engineering tool capabilities, Journal of Software Maintenance: Research and Practice, Volume 10, Issue 5, Pages: 305 – 331, John Wiley & Sons, Ltd, 1998.

[Beszédes et al., 1999] Beszédes, Á., Ferenc, R., Gyimóthy, T., Magyar, F., Márton, G., Tarkiainen, M., An evaluation of reverse engineering capabilities of the TDE/Columbus system, Technical Report, University of Szeged, 1999.

[Beszédes et al., 2005] Beszédes, Á., Ferenc, R., Gyimóthy, T., Columbus: A reverse engineering approach, Pre-Proceedings of the 13th Workshop on Software Technology and Engineering Practice, Budapest, Hungary, pages 93-96, September 24-25, 2005.

[Chikofski et al., 1990] Chikofski, E.J., Cross II, J.H., Reverse engineering and design recovery: A Taxonomy, IEEE Software, Volume 7, Issue 1, Pages: 13-17, January 1990.

[Demeyer et al., 1999] Demeyer, S., Ducasse, S., Lanza, M., A hybrid reverse engineering approach combining metrics and program visualization, In the 6th Working Conference on Reverse Engineering, 1999.

[Ducasse, 2003] Ducasse, S., Reengineering object oriented applications, Thesis, University of Bern, Switzerland, 2003.

[Ferenc et al., 2002] Ferenc, R., Beszédes, Á., Gyimóthy, T., Tarkiainen, M. , Columbus – Reverse engineering tool and schema for C++, International conference on Software Maintenance, pages: 172-181, 2002.

[Ferenc et al., 2001] Ferenc, R., Beszédes, Á., Magyar, F., Tarkiainen, M., Kiss, Á., Columbus-Tool for reverse engineering large object oriented software systems, 2001.

[Gall et al., 1996] Gall, H., Jazayeri, M., Klösch, R., Lugmayr, W., Trausmuth, G., Architecture recovery in ARES, Joint proceedings of the second international software architecture workshop (ISAW-2) and international workshop on multiple perspectives in software development (Viewpoints '96) on SIGSOFT '96 workshops, Pages:111-115, 1996.

[Harris et al., 1995] Harris, D.R., Reubenstein, H.B., Yeh, A.S., Reverse engineering to architectural level, International Conference on Software Engineering, Proceedings of the 17th international conference on Software engineering , Pages: 186 - 195, 1995.

[Imagix 4D User Guide, 2008] Imagix 4D User Guide, 2008.

[Imagix 4D webpage, 2008] Imagix 4D webpage, http://www.imagix.com/products/products.html , March 2008.

[Jarzabek et al., 1998] Jarzabek, S., Wang, G., Model based - design of reverse engineering tools, Journal of Software Maintenance: Research and Practice, Volume 10, Issue 5, Pages: 353-380, 1998.

[Jha et al., 2004] Jha, M., Maheshwaki, P., Phan, T.K.A., A Comparison of four software architecture reconstruction toolkits, 2004.

[Klösch, 1996] Klösch, R.R., Reverse Engineering: Why and how to reverse engineer software, 1996.

[Knodel and Pinzger, 2003] Knodel, J., Pinzger, M., Improving fact extraction of framework-based software systems, 10th Working Conference on Reverse Engineering, WCRE 2003.

[Koschke, 2005] Koschke, R., What architects should know about reverse engineering and reengineering, 5th working conference on Software Architecture, Page(s):4 – 10, IEEE, 2005.

[Lanza, 2003] Lanza, M., CodeCrawler-lessons learned in building a software visualization tool, In Proceedings of CSMR 2003, 2003.

[Louzado, 2005] Louzado, N., A reverse engineering tool for the analysis and comprehension of source code, May, 2005.

[Mendelzon and Sametinger, 1995] Mendelzon, A., Sametinger, J., Reverse engineering by visualizing and querying, Software Concepts and Tools, Vol. 16/4, pp. 170-182, December 1995.

[Müller et al., 2000] Müller, H.A., Wong, K., Tilley, S.R., Storey, M.A., Jahnke, J.H., Smith, D.B., Reverse Engineering: A road map, Proceedings of the Conference on The Future of Software Engineering, International Conference on Software Engineering, Pages:47-60, 2000.

[Müller et al., 1993] Müller, H.A., Wong, K., Tilley, S.R., Understanding software systems using reverse engineering technology, Proceedings of the 1993 conference of the Centre for Advanced Studies on Collaborat ive research: software engineering - Volume 1, Pages:217-226, 1993.

[Nelson, 1996] Nelson, M.L., A survey of reverse engineering and program comprehension, April 1996.

[Quigley et al., 2000] Quigley, A.J., Postema, M., Schmidt, H., ReVis: Reverse engineering by clustering and visual object classification.

[Rigi User's manual, 1998] Rigi User's manual version 5.4.4, June 1998.

[Rigi webpage, 2008] Rigi webpage, http://www.rigi.csc.uvic.ca/, March 2008.

[Setup and user's guide to Columbus/CAN, 2003] Setup and user's guide to Columbus/CAN, 2003.

[Storey et al., 2002] Storey, M.D., Sim, S.E., Wong, K., A collaborative demonstration of reverse engineering tools, ACM SIGAPP Applied Computing Review, Volume 10, Issue 1, Pages: 18 – 25, 2002.

[Storey et al., 1997] Storey, M.D., Wong, K., Müller, H.A., Rigi: A visualization environment for reverse engineering, Proceedings of the 19 th International Conference on Software Engineering, Page(s):606 – 607, May 1997.

[Storey et al., 1996] Storey, M.D., Wong K., Müller, H.A., Fong, P., Hooper D., Hopkins K., On designing an experiment to evaluate a reverse engineering tool, Proceedings of the Third Working Conference on Reverse Engineering, Page(s):31 – 40, November 1996.

[Systä, 1999] Systä, T., On the relationships between static and dynamic models in reverse engineering java software, Proceedings. Sixth Working Conference on Reverse Engineering, Page(s):304 – 313, October 1999.

[Systä et al., 2001] Systä, T., Koskimies, K., Mül ler, H., Simba - an environment for reverse engineering Java software systems, 2001.

[Tilley, 1998] Tilley, S., A reverse engineering environment framework, Technical Report CMU/SEI, April 1998.

[Tilley et al., 1996] Tilley, S.R., Paul, S., Towards a frame work for program understanding, Fourth workshop on program comprehension, pages: 19 -28, 1996.

[Tip, 1994] Tip, F., A survey of program slicing techniques, CWI (Centre for Mathematics and Computer Science), Amsterdam, The Netherlands, 1994.

[Tonella et al., 2007] Tonella, P., Torchiano, M., Bois, B.D., Systä T., Empirical studies in reverse engineering: state of the art and future trends, Empirical Software Engineering, Volume 12, Issue 5, Pages: 551 – 571, 2007.

[Understand webpage, 2008] Understand webpage , http://www.scitools.com/products/understand/ , July 2008.

[Understand User guide, 2008] Understand User guide and reference manual, version 2.0, 2008.

[Zayour] Zayour, I., Reverse Engineering: A Cognitive approach, a case study and a tool, Thesis, University of Ottawa.

Appendix

Available reverse engineering tools for C++ software

Tool name	Platform	Tool URL	Comments
Rigi	Windows, Linux, Solaris, etc	http://www.rigi.csc.uvic.ca/	Free tool
Columbus / CAN	Windows	http://www.frontendart.com./products_col.php	
Imagix 4D	Windows, Linux, Solaris, etc	http://www.imagix.com/products/products.html	
CodeCrawler	Every major platfrom	http://www.inf.unisi.ch/faculty/lanza/codecrawler.html	Free, language independent tool
Understand	Windows, Linux, Solaris, etc	http://www.scitools.com/products/understand/cpp/product.php	
Visustin	Windows	http://www.aivosto.com/visustin.html	
Codesurfer	Windows, Linux	http://www.grammatech.com/products/codesurfer/overview.html	C/C++ static source code analysis tool
Insight	Windows, Linux	http://www.klocwork.com/products/insight.asp	Static code analysis tool for C/C++ and Java
With Class	Windows	http://microgold.com/	
Rational Rose	Windows	http://www-306.ibm.com/software/awdtools/developer/rose/index.html	
SNiFF+	Unix	http://www.freedownloadscenter.com/Programming/C_and_C___Tools_and_Components/SNiFF_.html	
Crystal Flow for C++	Windows	http://www.sgvsarc.com/product_crystalflow.htm	Flowcharts from C++ source code
Source Navigator	Linux	http://sourcenav.sourceforge.net/	Open source tool
Code Visualizer	Windows	http://www.codedrawer.com/products/codedrawer.html	

Table 3: Available reverse engineering tools for C++ applications