GENERATING UML CLASS DIAGRAM FROM SOURCE CODES USING MULIT-THREADING TECHNIQUE

SAIF KHALID ABDULLAH

A dissertation submitted in
partial fulfilment of the requirement for the award of the
Degree of Master of Computer Science (Software Engineering)

Faculty of Computer Science and Information Technology
Universiti Tun Hussein Onn Malaysia

APRIL 2015

To my beloved father and mother

This dissertation is dedicated to my father, who taught me that the best kind of knowledge to have is that which is learned for its own sake. It is also dedicated to my beloved mother, who taught me that even the largest task can be accomplished if it is done one step at a time.

“Thank you for your love and support “

ACKNOWLEDGMENT

First and foremost, I would like to thank the almighty God (ALLAH S.W.T) for all the abilities and opportunities He provided me through all my life and His blessings that enables me to do this research.

It is the greatest honor to be able to work under supervision of Prof. Dr. Rosziati Ibrahim. I am grateful to be one of the luckiest persons who had a chance to work with her. I am thankful and gratified for all of her help, assistance, inspiration and guidance on the all aspects beside her patience and understanding

I wish to express my sincere gratitude to everyone who contributed to the successful completion of my study. I would like to express my gratitude to Universiti Tun Hussein Onn Malaysia (UTHM) for all support through my master.

ABSTRACT

The traditional Software Development Life Cycle (SDLC) often includes four phases: analysis, design, implementation, and testing. Reverse engineering is the process of moving back those phases by analyzing the software system and then representing it at the higher levels of abstractions. The reverse engineering software process generates high level information from the implementation phase. This information includes generating several diagrams and specification documents that describe the implemented software. The UML class diagram represent a valuable source of information even after the delivery of the software. Class diagram extraction can be done either from software's source code, or from the executable file. In the case of source code, a review of the current tools shows that many researchers have been extracting the UML class diagram from an object-oriented source code based on the sequential processing approach. In this research, a proposed approach for extracting a class diagram from the source code is presented. The proposed approach relies on multi-threading technique in the class diagram extraction which is representing the parallel processing. The motivation behind using multi-threading technique is that, it gives an advantage of faster processing to any software because the threads of the program naturally lend themselves to truly concurrent execution. In this research, a class diagram extraction using multi-threading technique is designed and implemented using the C# programming language. The implemented approach is tested on three case studies that contain several types of entities and relationships between them. Testing results show that the time needed to extract class diagram using multi-threading technique for the tested three cases is less than the time needed in extracting the same class diagram without using multi-threading technique.

ABSTRAK

Kitar Hayat Pembangunan Sistem (KHPS) terdiri daripada empat fasa: analisis, reka bentuk, pelaksanaan, dan pengujian. Kejuruteraan balikan adalah proses pergerakan ke mengundur kesemua fasa dengan menganalisa sistem perisian dan kemudiannya mewakilkan ia pada abstraksi tahap tinggi. Proses kejuruteraan balikan perisian menjana maklumat tahap tinggi daripada fasa pelaksanaan. Maklumat ini merangkumi penjanaan sesetengah rajah dan spesifikasi dokumen yang menggambarkan perisian yang dilaksanakan. rajah kelas mewakili sumber maklumat yang paling berharga walaupun selepas penghantaran perisian. Pengekstrakan rajah kelas boleh dibuat daripada kod sumber perisian atau daripada fail pelaksanaan. Dalam kes kod sumber, kajian peralatan semasa menunjukkan bahawa kebanyakan penyelidik telah mengekstrak rajah kelas UML daripada kod sumber berorientasikan objek berasaskan pendekatan pemprosesan berjujukan. Dalam kajian ini, pendekatan yang dicadangkan untuk mengekstrak rajah kelas daripada kod sumber dibentangkan. Pendekatan yang dicadangkan bersandarkan kepada teknik multi-threading dalam pengekstrakan rajah kelas yang mewakili pemprosesan selari. Motivasi di sebalik menggunakan teknik *multi-threading* adalah kelebihan pemprosesan lebih cepat terhadap perisian kerana bebenang program secara semulajadi meminjamkan dirinya kepada pelaksanaan serentak. Dalam kajian ini, pengakstrakan rajah kelas menggunakan teknik *multi-threading* direkabentuk dan dilaksanakan menggunakan Bahasa pengaturcaraan C#. Pendekatan yang dilaksanakan diuji pada tiga kajian kes yang mengandungi beberapa jenis entiti dan hubungan antara mereka. Hasil pengujian menunjukkan masa yang diperlukan untuk mengekstrak rajah kelas menggunakan teknik *multi-threading* bagi tiga kajian kes yang diuji adalah kurang daripada masa yang diperlukan untuk mengekstrak rajah kelas yang sama tanpa menggunakan teknik *multi-threading*.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF APPENDICES

CHAPTER 1

INTRODUCTION

1.1 Background of research

Software maintenance is the last phase in the life cycle of a software development process which often includes the following phases: requirement specification, analysis, design, implementation, testing, deployment and maintenance (Dennis, *el at.*, 2006). However, this phase plays an important role because software maintenance activities ensure that a software system still works well without errors in new environments after it is released. According to Doan (2008), common maintenance activities include fixing bugs, adapting the system to a new environment, adding new features to the system to satisfy new requirements from the client, and updating documentation for the system. In order to do these tasks, software maintainers must understand the structure or architecture of the system. However, it is a hard task for them in case some changes happened in the structure of the system, which makes the system different from its original version. In some cases, system documentation 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 (Doan, 2008).

In the case of source code reuse, if some parts of a new software system can be reused from existing systems, software developer will save a large amount of money and effort in developing it (Doan, 2008). In order to reuse the source code, software developers must realize the structure and architecture of the system and then understand clearly their features and functions.

Reverse engineering tools are very useful in the above cases. The term reverse engineering was defined by Chikofsky & Cross (1990) as the process of analyzing a subject system to (i) identify the system's components and their interrelationships and (ii) generate representations of the system in another form or at a higher level of abstraction. It is an activity that takes place frequently, for example, when understanding a system before making a change; when migrating a software system from one platform to another; when transforming source code from one object model to another; and when refactoring a set of classes to satisfy new requirements (Canfora & Penta, 2007).

According to Tonella & Potrich (2001) reverse engineering tools provide useful high level information about the system being maintained. Their output diagrams can support the program in understanding activities, drive refactoring, and restructuring interventions, and also employed to assess the traceability of the design into the code. Therefore, it is important that the representations recovered from the code to be accurate, i.e., exploit all static information present in the code in order to reverse engineer entities and relations.

Enhancements can be easily done if the modeling diagrams are done during the initial diagram generation. Unfortunately, when the software is delivered, design diagrams are not packed with it. There are a large number of tools that are incorporated with reverse engineering modules (Nagappan, 2008). The most commonly implemented reverse engineering module is the reverse engineering of the codes (Nagappan, 2008).

A thread is the smallest sequence of programmed instructions that can be managed independently by an operating system scheduler (Butenhof, 1997). According to Justia (2011), multi-threading is the ability of a program or an operating system process to manage multiple requests by the same user without having to have multiple copies of the programming running on the computer. Multi-threading paradigm has become more popular as efforts to further exploit instruction level parallelism have stalled since the late 1990s (Justia, 2011). This allowed the concept of throughput computing to re-emerge to prominence from the more specialized field of transaction processing.

1.2 Research motivations

Software engineering has undergone a paradigm shift as the size of the software systems deployed increased dramatically and businesses began to rely increasingly on computers and information systems (Ramasubbu, 2001). A substantial portion of the software development effort is spent on maintaining existing systems rather than developing new ones (Rugaber, 1992). An estimated 50% to 80% of the time and material involved in software development is devoted to maintenance of existing code (Boehm, 1991). Crucial to the maintenance of existing systems is the task of program comprehension, an emerging area in software engineering. About 47% of the time is spent on enhancements to existing programs and 62% of that spent on program corrections involve program comprehension tasks like reading the documentation, scanning the source code, and understanding the changes to be made (Fjeldstad & Hamlen, 2001).

Software development as mentioned above is a growing field. However, developing software from scratch is no longer a situation faced by the developer. The challenge faced currently is how to use the minimum information about existing software and further enhance it to become a powerful tool (Nagappan, 2008).

Since the paradigm shift, developers who have embarked on the idea of enhancing any software are often faced with the problem of how to gather the initial requirements on which the existing software was built upon (Nagappan, 2008). Documentation that is often used to aid this process would be the user manual. However, user manuals only show how to use the system for the system user and not from the developer's perspective. Design documentations are often not enclosed together with the software due to security reasons. UML models are used to document user requirements and design documentation. One of the most important models used is the class diagram. Class diagram describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects (Nagappan, 2008).

According to Barnes, *et al.*, (2012) the advantage of a multithreaded program is that it allows the program to operate faster on computer systems. This is because the threads of the program naturally lend themselves to truly concurrent execution.

Many researchers have developed techniques and tools of reverse engineering from source code to class diagram such as (Ibrahim, R. & Yong, T.K., 2008), where they developed ReSet tool which is implemented using C++ programming language. The main objectives of developing this tool is being able to detect the necessary tokens from the syntax of the program source codes and generate the class diagram automatically based on the detected tokens. Another tool named ForUML is proposed by Aziz, *et al.*, (2013). ForUML is a tool that extracts UML class diagrams from Fortran code. ForUML can produce an XMI document that describes the UML Class Diagrams. While Jain, *et al.*, (2010) developed a reverse engineering method to automate the extraction of DFDs, CFDs, and class diagrams from any legacy C++ code. The extracted information is classified as structural, behavioral and constraint rules through which such information can be produced. And also many tools are developed such as in (Matzko, *et al.*, 2002, Sutton & Maletic, 2007, Keschenau, 2006, Tonella, 2005). However, none of these researches have used multi-threading technique to extract UML class diagram from the source code. Therefore, this research uses multi-threading technique to improve the efficiency of UML class diagram extracted from source code.

1.3 Objectives

The objectives of this research work are as follows:

- To design an approach that generates class diagram from object-oriented source code using multi-threading technique.
- To implement the proposed approach using C# programming language.
- To test the proposed approach on C# source code and compare it with generating a class diagram without using multi-threading technique for its efficiency in terms of time.

1.4 Scope of research

The main area of concentration in this research is the part on reverse engineering that is pertaining to generating class diagram from source code. The reverse engineering

concept here is explained in terms of transformation of object oriented source codes to UML diagrams using the suggested approach. The application scope of this approach will be the C# source code only. The main focus will be on using asynchronous threading. Asynchronous concept in C# programming language explained the running of two or more operations in different contexts (thread) so they can run concurrently and do not block each other (Mazhou, 2010). The application accepts codes that are free from any syntax errors. The parser that will be built according to the suggested approach is limited only to extracting class diagram, not compiling the source code. The input will be source code of C# language and the output will be a UML class diagram. Four relationships between classes and interfaces will be extracted: Generalizations, realizations, association, and dependency. The proposed approach's aim is to compare the time needed in generating a class diagram with and without using the multi-threading technique. The proposed approach is applied on three case studies in order to prove its validity. The three case studies are C# programs that contain several code files. Each code file contains a set of classes and interfaces. The time of execution is calculated for each case study and then listed in the testing results table.

1.5 Expected outcomes

The main aim of this research is to prove that using multi-threading technique in generating class diagram from source code is more efficient than generating it without using the multi-threading technique. The basic criterion of comparison is time. A tool will be developed to compare the time needed in generating class diagram with and without using multi-threading technique. This research outcome will be a tool that generates a class diagram from source code in two ways: i) Using multi-threading technique, ii) Without using multi-threading technique. The developed tool generates a class diagram that contains the following items: classes, interfaces, relationships, class attributes, and class operations. The developed tool still needs some enhancements in order to be able to extract class diagram using all possible rules of code writing.

1.6 Chapter summary

In this chapter the overall aim of the research which is generating UML class diagram from an implementation phase using reverse engineering is explained. Further illustrate on how the aims can be achieved, objectives are identified. The scope and the limitation in this chapter clearly narrow the broad area of research. This chapter also gives an introduction and a brief overview of the reverse engineering concept and its advantages. Followed by the expected outcomes and the organization of the dissertation. The next chapter will cover the literatures related to this research.

1.7 Dissertation outline

The rest of this dissertation is organized into the following chapters. Chapter 2 presents a review of several aspects that are related to this research, starting from general ideas of reverse engineering to UML and class diagram. After that, a description of C# programming language is introduced. This is followed by a description of multi-threading technique and how to apply it practically. Finally, some of the related works of this research are presented. Chapter 3 contains a description of the proposed methodology that will be followed and used in order to achieve this research objective. This chapter starts with an overall view of the proposed methodology. After that, a description of the methodology in brief details are presented. The methodology is composed of a set of vital steps. Each step takes the result of the previous step and provides a new input for the next step. While in Chapter 4, design and implementation phases of this research are introduced. The design part will be about how to generate a class diagram in general. Then the proposed work is presented. The proposed work consists of mainly two parts: generating a class diagram from source code using multi-threading technique, and generating a class diagram without using multi-threading technique. After design, the last two parts of this research are presented, namely tool implementation and testing. In the first section which is implementation, a detailed description of the proposed approach is to be presented. Implementation explanation contains also some important parts of the written code. The second section will be about tool testing.

Testing is done on three case studies that will be explained in the testing section. The main concentration in testing will be the time of execution. This means processing time that is needed to generate a class diagram with and without using multi-threading technique. Chapter 5 provides a summary of this research, which will be presented along with possible future development.

CHAPTER 2

LITERATURE REVIEW

2.1 Introduction

This chapter covers the literatures related to this research. First of all, an overview of reverse engineering and its derivatives with its main activities are presented. After that, the description of UML as a prominent modelling language is given. Then, the class diagram and its relationship types are discussed. Next, an overview of C# language is given. This is followed by a description of multi-threading technique and how to apply it practically. After that, some of the related works of this research are presented. Finally, summarizes the topics discussed in this chapter.

2.2 Reverse engineering derivatives

In order to understand clearly about reverse engineering, one must distinguish it from other terms such as restructuring and reengineering. These processes are described in Figure 2.1. In the following sections, a description of each term is presented.

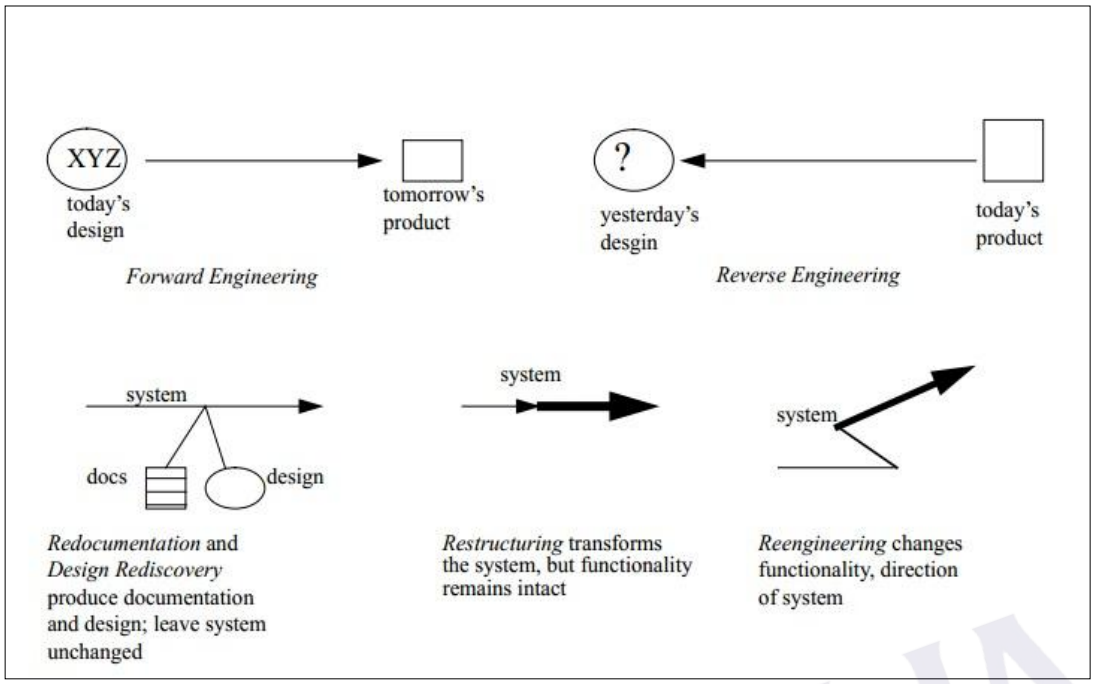

Figure 2.1: Reverse engineering derivatives (Nelson, 1996)

2.2.1 Reverse engineering

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 often includes four phases: analysis, design, implementation, and testing. Through those phases, software is developed from the high level of abstraction (architecture) to the lowest level of abstraction (source code). Therefore, reverse engineering is the process which analyzes software system and then represents it at the higher levels of abstractions. The following definitions which is given by Chikofsky & Cross (1990) is widely used: "Reverse engineering is the process of analyzing a subject system to identify the system's components and their interrelationships."

2.2.2 Restructuring

Restructuring is the transformation from one representation form to another form within the same abstraction level (Chikofsky & Cross 1990). An example would be to modify the source code in order to make the structure of the 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 intended by the software engineers, but still at 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 generates changes in the structure of the system, while reverse engineering only examines the structure of the system and does not make any changes in the system.

2.2.3 Reengineering

Reengineering is the examination, alteration, and modification of the system in order to regenerate a new system with new functions in another representation form (Chikofsky & Cross 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 a higher level of abstraction. At that time, some changes were generated 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 generated. This phase follows the steps in forward engineering. Hence, reengineering generates 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 replication.

2.3 Reverse engineering activities

According to Tilley (1998), the reverse engineering process includes three main activities: data gathering, knowledge management, and information exploration.

2.3.1 Data gathering

One cannot understand about the structure of a software system at higher levels of abstraction without having the necessary information pertaining to the subject.

Therefore, data gathering is often the first step, where several types of data about the system are gathered such as the source code, comments in source code, documentation about the system, and experts' comments. Three techniques of data gathering which are widely used are: system examination, document scanning, and experience capture (Tilley, 1998).

2.3.1.1 System examination

System examination is often classified into two constricting ways: static examination and dynamic examination. Static examination concentrates on analyzing the source code. A source code parser is often used to analyze the source code and then transfers it to abstract syntax trees (Bellay & Gall, 1998). In contrast, 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 the knowledge about the interactions between components in the system, types of messages and protocols used, and the external recourses used by the system (Tilley, 1998).

2.3.1.2 Documents scanning

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 the experts (Tilley, 1998).

2.3.1.3 Experience capturing

Experience capturing is the approach to obtain knowledge about the system by interviewing the people who developed the system. The knowledge is very useful in

understanding the system. However, it is difficult to find out those that developed the system (Tilley, 1998).

2.3.2 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 namely knowledge organization, knowledge discovery, and knowledge evolution (Tilley, 1998).

2.3.3 Information exploration

According to Tilley (1998), the majority of program understanding takes place during information exploration, and it is arguably the most important of the three canonical reverse engineering activities. Data gathering is required to begin the reverse-engineering process. Knowledge management is needed to structure the data into a conceptual model of the application domain. But the key to increased comprehension is exploration because it facilitates the iterative refinement of the hypotheses. The process of information exploration includes three activities: navigation, analysis, and presentation (Tilley, 1998).

2.4 Unified Modelling Language (UML)

UML is defined by (OMG, 2008) as: "a graphical language for visualizing, specifying, constructing and documenting the artifacts of software systems". UML was originally derived from object modeling languages of three leading object-oriented methods: Booch, Object Modeling Technique (OMT), and Object-Oriented Software Engineering (OOSE). It is more compatible to be used to model object-oriented software systems.

Object Management Group (OMG) has approved UML in November 1997 as the standard notation for object-oriented analysis and design, and it became the industry standard for modeling objects and components. At the end of 2000, the OMG has issued a Request For Information (RFI) with regard to UML 2.0.

REFERENCES

Agile. (2014). *UML 2 Class Diagrams: An Agile Introduction*. Retrieved on 07/03/2014,from: http://www.agilemodeling.com/artifacts/classDiagram.htm.

Aziz, N., Karla Morris and Salvatore Filippone (2013). Extracting UML class diagrams from object-oriented Fortran: ForUML. *Proceedings of the 1st International Workshop on Software Engineering for High Performance Computing in Computational Science and Engineering (SE-HPCCSE '13).* New York: ACM, pp 9-16.

Barnes, A., Ryan Fernando, Kasuni Mettananda and Roshan Ragel (2012). Improving the Throughput of the AES Algorithm with Multi core Processors. *Proceeding of the 7th International Conference on Industrial and Information Systems (ICIIS)*. Chennai: IEEE.pp.1-6.

Bell, and James R. (2003). Threaded code. *Communications of the ACM*, ACM, 16 (6), pp. 370–372.

Bellay, B. and Gall, H.(1998). An evolution of reverse engineering tool capabilities. *Journal of Software Maintenance: Research and practice*, 10(5), pp. 305-33.

Britannica (2010). *Sequence Programming Britannica Online Encyclopedia*. Retrieved on 13/05/2014 from: http://www.britannica.com/EBchecked/topic/1086517/sequence.

Boehm, B.W.(1991). *Software Engineering Economics*. 1st edition.USA: Prentice Hall.

Butenhof, D.R. (1997). *Programming with POSIX Threads*. 1st edition. Boston, USA: Addison-Wesley.

Canfora, G. and Penta, M. (2007). New Frontiers of Reverse Engineering. *Future of Software Engineering (FOSE '07)*. Minneapolis: IEEE. pp. 326-341.

Chikofsky, E. and Cross, J. I. (1990). Reverse engineering and design recovery: A taxonomy. *IEEE Software, 7(1)*, pp. 13-17.

Dennis, A., Wixom, B.H. and Roth, R.M. (2006). *Systems Analysis and Design*. 3rd edition. Hoboken: John Wiley & Sons, Inc.

Doan, T.(2008). *An evaluation of four reverse engineering tools for C++ applications*. University of Tampere: Master's Dissertation.

Dobing, B. and Parsons, J. (2006). *How UML is used*. Communications of the ACM - Two decades of the language-action perspective, ACM, 49(5).pp. 109-114.

Ertl, A. *what is threaded code*. Retrieved on 11/03/2014 from: http://www.complang.tuwien.ac.at/forth/threaded-code.html#what

Fjeldstad, R.K., and Hamlen W.T. (2001). Application Program Maintenance Study: Report to Our Respondents. *Tutorial on Software Maintenance*. IEEE Computer Society Press.pp. 13 – 30.

Goetz, B. (2002). *Java theory and practice: Thread pools and work queues*. IBM DeveloperWorks. Retrieved on 12/03/2014 from: http://www.ibm.com/developerworks/java/library/j-jtp0730/index.html.

Grossman, M., Aronson, J. E. and McCarthy, R. V. (2005). Does UML make the grade? Insights from the software development community. *Information and Software Technology*, *47(6)*, pp. 383-397.

Harvey, B.,Wright, M. (1999). *Simply scheme: introducing computer science*. 2nd edition. California: MIT Press.

Ibrahim, N.(2013). *An Enhanced UML Consisteccy Cecker Using Logical Approach*. Universiti Tun Hussein Onn Malaysia: Ph.D. Dissertation.

Ibrahim, R. & Yong, T.K. (2008). ReSeT: Reverse Engineering System Requirements Tool. *World Academy of Science, 14(4)*, pp.238-241.

Jain, A., Sooner, S, and Holkar, A. (2010). Reverse engineering: Extracting information from C++ code. *Proceedings of the 2nd International Conference on Software Technology and Engineering (ICSTE)*. San Juan: IEEE. pp. 154 - 158 .

Justia (2011). *Obfuscated hardware multi-threading*. Retrieved on 31/03/2014, from http://patents.justia.com/patent/8621186.

Justia. (2013). *Multi-threading Computers*. Retrieved on 11/03/2014 from: http://patents.justia.com/patent/20140047219.

Keschenau, M. (2006). Reverse Engineering of UML Specifications from Java Programs. *Proceedings of the 19th companion annual conference on Object-oriented programming systems (OOPSLA '04).*New York:ACM,pp 326-327.

Matzko, S., Clarke, P. J., Gibbs, T. H., Malloy, B. A., Power, J. F., and Monahan, R. (2002). Reveal: a tool to reverse engineer class diagrams. *Proceedings of the Fortieth International Conference on Tools Pacific: Objects for internet,*

mobile and embedded applications (CRPIT'02). Australia: ACM, 10(4), pp.13-21.

Mazhou.(2010). What are actually synchronous/asynchronous operations (C# 5.0 Series). Retrieved on 05/07/2014, from http://www.codeproject.com/Articles/127660/What-are-actually-synchronous asynchronous-operati.

Mivule, K. (2011). *Difference between Sequential and Parallel Programming*. Retrieved on 12/05/2014 from: https://mivuletech.wordpress.com/2011/01/12/difference-between-sequential-and-parallel-programming/.

Miller, R. (2003). *Practical UML : A Hands-on Introduction for developer*. Retrieved on 05/03/2014, from http://dn.codegear.com/article/31863.

MSDN. (2014). *Microsoft Corporation*. Retrieved on 08/03/2014, from: http://msdn.microsoft.com/en-us/default.aspx.

Nagappan, S.D. (2008). *A reverse engineering uml modeling tool*. University of Malaya: Master's Dissertation.

Nelson, M.L. (1996). A survey of reverse engineering and program comprehension, *A Survey of Reverse Engineering and Program Comprehension*. Computing Research Repository - CORR.pp.1- 8.

Nishadha. (2012). Class Diagram Relationships in UML with Examples. Retrieved on 08/03/2014, from:http://generately.com/blog/diagrams/class-diagram-relationships/.

OMG (2008). Unified Modeling Language UML, http://www.omg.org/-spec/UML/, 2008. OMG Formally Released Versions of UML and ISO Released Versions of UML. p. 20,40, 116.Pages 212-229.

Ramasubbu, S. (2001). *Reverse Software Engineering Large Object Oriented Software Systems using the UML Notation*. Virginia Polytechnic Institute and State University: Master's Dissertation.

Rohitha. (2011). *Understanding UML Class Diagram Relationships*. Retrieved on 07/03/2014, from: http://generately.com/blog/diagrams/understanding-the-relationships-between-classes/.

Rugaber,S. (1992). Program Comprehension for Reverse Engineering. Workshop on AI and Automated Program Understanding.

Scott, W. A. (2009). *UML 2 Class Diagrams.* Retrieved on 07/03/2014, from

http://www.agilemodeling.com/artifacts/classDiagram.htm

Sparks, G. (2001). Database Modelling in UML. Retrieved on 06/03/2014, from http://www.methodsandtools.com/archive/archive.php?id=9.

Sutton, A. and Maletic, J.I. (2007). Recovering UML class models from C++: A detailed explanation. *Information and Software Technology*. 49(3), pp 212-229.

Thaara, S. (2002). *Thread Pool Pattern*. Retrieved on 12/03/2014 from: http://www.scribd.com/doc/240669342/Thread-Pool-Pattern.

Tilly, S.T.(1998). A reverse engineering environment framework. *Technical report CMU/SEI*, Hanscom: Software Engineering Institute.

Tonella, P. (2005). Reverse Engineering of Object Oriented Code. *Proceeding of the. 27th International Conference on Software Engineering ICSE 2005*.IEEE .pp.724- 725.

Tonella, P., and Potrich, A. (2001). Reverse engineering of the UML class diagram from C++ code in presence of weakly typed containers. *Proceeding of International Conference on Software Maintenance*. Florence: IEEE.pp. 376 – 385.

Tokhi, Mohammad, A. H. and Mohammad, H. S. (2003). Parallel computing for real-time signal processing and control, *Advanced textbooks in control and signal processing*.43(11),pp 1545-1568.

Vidgen, R. (2003). Requirements analysis and UML use cases and class diagrams. *Computing & Control Engineering Journal*.14(2).pp.12 - 17.

Vinita; Amita Jain and Devendra K. Tayal. (2008). On reverse engineering an object-oriented code into UML class diagrams incorporating extensible mechanisms. *ACM SIGSOFT Software Engineering Notes*. 33(5), pp.1-9.

Wiki. (2007). *Thread pool*. Retrieved on 12/03/2014 from: http://en.wikipedia.org/w/index.php?title=File:Thread_pool.svg.

Wiki. (2014). *Class diagram relationships examples*. Retrieved on 06/03/2014, from: http://en.wikipedia.org/wiki/Class_diagram

Xmipp (2010). *ParallelProgramming*. Retrieved on 12/05/2014 from: http://xmipp.cnb.csic.es/twiki/bin/view/Xmipp/ParallelProgramming.