UNIVERSITI PUTRA MALAYSIA

EXTRACTING OBJECT ORIENTED SOFTWARE ARCHITECTURE FROM C++ SOURCE CODE

ALI HUSSEIN A. MRESA

FSKTM 2000 6

EXTRACTING OBJECT ORIENTED SOFTWARE ARCHITECTURE FROM C++ SOURCE CODE

By

ALI HUSSEIN A. MRESA

Thesis Submitted in Fulfilment of the Requirements for the Degree of Master of Science in the Faculty of Computer Science and Information Technology Universiti Putra Malaysia

October 2000

To the Soul of My Sisters Al-forjania and Soad

To My Parents

Abstract of thesis presented to Senate of University Putra Malaysia in fulfilment of the requirements for the degree of Master Science.

EXTRACTING OBJECT-ORIENTED SOFTWARE ARCHITECTURE FROM C++ SOURCE CODE

By

ALI HUSSEIN A. MRESA

October 2000

Chairman: Abdul Azim Abdul Ghani, Ph.D.

Faculty: Computer Science and Information Technology

Software architecture strongly influences the ability to satisfy quality attributes such as modifiability, performance, and security. It is important to be able to analyse and extract information about that architecture. However, architectural documentation frequently does not exist, and when it does, it is often out of sync with the implemented system. In addition, it is not all that software development begins with a clean slate; systems are almost always constrained by the existing legacy code. As a consequence, there is a need to extract information from existing system implementations and reason architecturally about this information.

This research presents a reverse engineering tool VOO++ that will read an Object-Oriented C++ source code using UML notation in order to visualise its Class structure and the various relationships that may exist including, inheritance, aggregation, and dependency relationships based on the modified Cohen-Sutherland clipping algorithm.

The idea of clipping is reversed, instead of clipping inside the rectangle, the clipping is done out side the rectangle in terms of four directions (left, right, top, and bottom) and two points represent the centre point for each rectangle.

An Object-Oriented approach is used to design and implement the tool. Reverse engineering, design pattern, and graphics are the underlying techniques supplied. VOO++ aids an analyst in extracting, manipulating and interpreting the Object-Oriented static model information. By assisting in the reconstruction of static architectures from extracted information, VOO++ helps an analyst to redocument and understand architectures and discover the relationship between “as-implemented” and “as-designed” architectures.

Abstrak tesis yang dikemukakan kepada Senat Universiti Putra Malaysia bagi memenuhi keperluan untuk ijazah Master Sains.

PENGHASILAN SENI BINA PERISIAN BERASASKAN OBJEK DARI KOD SUMBER C++

Oleh

ALI HUSSEIN A. MRESA

Oktober 2000

Pengerusi: Dr. Abdul Azim Abd Ghani, Ph.D.

Fakulti: Sains dan Pengajian Alam Sekitar

Seni bina perisian sangat mempengaruhi keupayaan untuk memenuhi atribut kualiti seperti kebolehubahan, prestasi dan sekuriti. Adalah penting untuk mampu menganalisa dan menaakul mengenai seni bina tersebut. Walau bagaimanapun dokumentasi seni bina kadang kala wujud, dan bila ianya wujud, ianya tidak selaras dengan sistem yang diimplemen. Tambahan pula tidak semua pembangunan perisian bermula dengan keperluan baru sepenuhnya; sistem dikekang oleh kewujudan kod legasi. Akibatnya kita perlu mampu menghasilkan maklumat dari implementasi sistem yang sedia ada dan menaakul secara seni bina mengenai maklumat ini.

Penyelidikan ini mempersembahkan satu alatan kejuruteraan songsang VOO++ yang dapat membaca kod sumber berorientasi objek C++ dan menghasilkan notasi

UML untuk mengvisualisasi struktur kelas dan pelbagai hubungan yang mungkin wujud termasuk pewarisan, agregasi dan hubungan kebergantungan berdasarkan kepada algoritma perubahan kepitan Cohen-Sutherland. Idea kepitan disongsangkan, sebagai gantian kepada kepitan dalam segi empat, kepitan dilakukan di luar segi empat dalam rangkaian empat arah (kiri, kanan, atas, bawah) dan dua titik mewakili titik tengah untuk setiap empat segi.

Pendekatan berorientasikan objek digunakan untuk mereka bentuk dan mengimplemen alatan tersebut. Teknik kejuruteraan songsang, corak reka bentuk dan grafik digunakan sebagai asas. VOO++ membantu penganalisa dalam menghasil, memanipulasi dan menginterpretasi maklumat statik model berorientasikan objek. Dengan membantu dalam membina semula seni bina statik dari maklumat yang terhasil, VOO++ menolong penganalisa untuk mendokumen semula dan memahami seni bina dan mengetahui hubungan diantara seni bina "yang diimplemen" dan "yang direka bentuk".

ACKNOWLEGEMENTS

In the name of Allah, Most Gracious, Most Merciful

I would like to take this opportunity to convey my sincere thanks and deepest gratitude to my chairman supervisor Dr. Abdul Azim Abdul Ghani for his advises, comments, suggestions, help, and invaluable guidance, and fruitful discussions throughout my research.

I am also indebted to Dr. Ramlan Mahmod and Dr. Md. Nasir sulaiman, members of the supervising committee, for their technical support, suggestions and insights are priceless.

I am greatly indebted to the Libyan Ministry of Education for financial support during my study. The contribution of the people at the Libyan Bureau is highly appreciated.

I owe the biggest debt to Engineering Academy and my family for their assistance and support. They are the biggest contributors to my success.

Finally, special mention must be made of all my friends at faculty of computer science and information technology without their knowledge this work would not have been done. These people deserve to be recognised and so I want to do, but I am afraid if I do, I will definitely miss some names.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

> "If a project has not achieved a system architecture, including its rationale, the project should not proceed to full-scale system development. Specifying the architecture as a deliverable enables its use throughout the development and maintenance process" (Boehm, 1995).

1.1 Background

Architectural design has always played a strong role in determining the success of complex software-based systems, the choice of an appropriate architecture can lead to a product that satisfies its requirements and is easily modified as new requirements present themselves, while an inappropriate architecture can be disastrous (Garlan, 1997; Buxton & McDermid, 1991).

Despite its importance to software systems engineers, the practice of architectural design has been largely ad hoc, and informal. As a result, architectural designs are often poorly understood by developers; architectural choices are based more on default than solid engineering principles; architectural designs cannot be analysed for consistency or completeness; architectural constraints assumed in the initial design are not enforced as a system evolves; and there are virtually no tools to help the architectural designers with their tasks (Garlan, 1997).

1.2 Software Architecture

Software architecture concerns the structures of large software systems. The architectural view of a system is an abstract view that distils away details of implementation, algorithm, and data representation and concentrates on the behaviour and interaction of "black-box" components. Software architecture is developed as the first step toward designing a system that has a collection of desired properties. Also the product of software design activities are the definition of the software architecture specification.

1.3 Importance of Software Architecture

Fundamentally, there are three reasons why software architecture is important, as follows (Bass *et al.*, 1998).

- Communication among stakeholders. Software architecture represents common high-level abstraction of a system that most if not all of the system's stakeholders can use as a basis for creating mutual understanding, forming consensus, and communicating with each other.

- Early design decisions. Software architecture represents the manifestation of the earliest design decisions about a system, and these early bindings carry weight far out of proportion to their individual gravity with respect to the system's remaining development, its deployment, and its maintenance life. It is also the earliest point at which the system to be built can be analysed.
- Transferable abstraction of a system. Software architecture constitutes a relatively small, intellectually graspable model for how a system is structured and how its components work together this model is transferable across systems; in particular, it can be applied to other systems exhibiting similar requirements and can promote large-scale reuse.

1.4 Software Architecture Issues

The formal study of software architecture has been a significant addition to the software-engineering repertoire in the 1990s. It has promised much to designers and developers: help with the high-level design of complex systems. Early analysis of high-level designs; particularly with respect to their satisfaction of quality attributes such as modifiability, security, and performance; higher level reuse such as that of designs and enhanced stakeholders communication (Garlan, 1993). These benefits seem enticing. However, much of the promise of software architecture has as yet gone unfulfilled. Some of the problems simply stem from the fact that architectures are seldom documented properly (Kazman & Carriere, 1997) where:

- Many systems have no documented architecture at all. (All systems have an architecture, but frequently it is not explicitly known or recorded by the developers and therefore evolves in an ad hoc fashion.)
- Architectures are represented in such a way that the relationship between the architectural representation and the actual system, particularly its source code, is unclear.
- In systems that do have properly documented architectures, the architectural representations are frequently out of sync with the actual system, because maintenance of the system occurs without a similar effort to maintain the architectural representation.
- There is little completely new development. Development is typically constrained by compatibility with, or use of, legacy systems. And it is rare that such systems have an accurately documented architecture. Because of these issues, a serious problem in assessing architectural conformance arising, which makes system understanding and maintaining unbearable. Mismatch between "as designed" and "as implemented" system architectures cause much of the value of having an architecture is lost.

In addition, when a system enters the maintenance phase of its life cycle, it may sustain modifications that alter its architecture. Hence, a second problem arises: how to ensure that maintenance operations are not eroding the architecture, breaking down abstractions, bridging layers, compromising information hiding, and so forth?

All of these are manifestations of two underlying causes. The first is that a system does not have "an architecture". It has many: its runtime relationships, data flows, control flows, code structure, and so on. The second, more serious, cause is that the architecture that is represented in a system's documentation may not coincide with any of these views.

1.5 Research Objectives

With reference to the software architecture issues the research is aiming to develop a prototype reverse-engineering tool base on an Object-Oriented approach with reverse engineering underlying. The proposed tool called VOO++ (**V**isualisation of **O**bject **O**riented Architecture from C++ source code) that helps the stakeholders to:

- Re-document Object-Oriented software architecture using an Object-Oriented reverse engineering tool, which implies using standard notations UML (unified modelling language) to facilitate the communications between project teams. As a result of documentation, the relationship between "as designed" and "as implemented" system architecture is presented, which is provided that the software under analysis should be documented during its development process.

However, the research undertaken will cover the following: