INTRODUCING CONTROL AND STRUCTURE IN SOFTWARE PROTOTYPING

MD. MAHBUBUR RAHIM

FSAS 1992 1

INTRODUCING CONTROL AND STRUCTURE IN SOFTWARE PROTOTYPING

By

MD. MAHBUBUR RAHIM

Thesis Submitted in Fulfilment of the Requirements for the Degree of Master of Science in the Faculty of Science and Environmental Studies
Universiti Pertanian Malaysia

March 1992

Dedicated in the name of the all merciful Almighty Allah.

ACKNOWLEDGEMENTS

The author wishes to express his deepest gratitude to Dr. Abu Talib Othman, Chairman of the supervisory committee, for his constant guidance, assistance and encouragement during the entire course of the study.

The author is extremely thankful to co-supervisor Mohd. Hasan Selamat, Lecturer, Department of Computer Science, UPM, for his invaluable advices and comments during the various stages of the study. His keen interest in this topic has always been a source of inspiration to the author.

The author also wishes to express his thanks to co-supervisors Mrs. Fatimah Ahmad and Md. Nasir Sulaiman, Lecturers, Department of Computer Science, for their kind assistance and scholarly guidance.

The author further likes to express especial thanks to B.A.A. Mustafi, for going through the manuscript from time to time.

The author also acknowledges the co-operation of all the Bangladeshi students in UPM and UKM that made his stay in Malaysia enjoyable and memorable.

The author further likes to record special appreciation to his beloved wife Shaheen Akhter for typing some parts of the thesis and for continuous morale support and encouragement.

Last but by no means the least, the author likes to express heartful indebtedness to his parents, brothers and sisters, parents-in-law, brother-in-law, sister-in-law and other members of his family who have supported and encouraged him during the entire period of the study.

TABLE OF CONTENTS

LIST OF TABLES

Page

LIST OF FIGURES

Page

Page

LIST OF ABBREVIATIONS

ACD	:	Action Chart Diagram
CASE	:	Computer Assisted Software Engineering
CBIS	:	Computer-Based Information Systems
DBMS	:	Data Base Management Systems
DFD	:	Data Flow Diagrammer
DMD	:	Data Model Diagrammer
E/R	:	Entity Relationship
FSAS	:	Faculty of Science and Environmental Studies
I-CASE	:	Integrated Computer Assisted Software Engineering
IS	:	Information Systems
LAN	:	Local Area Network
MIS	:	Management Information Systems
PA	:	Penasihat Akademik
POSE	:	Picture Oriented Software Engineering
SCD	:	Structure Chart Diagrammer
SDLC	:	Systems Development Life Cycle
SP	:	Structured Prototyping
ST	:	Structured Transitions
UPM	:	Universiti Pertanian Malaysia

Abstract of thesis submitted to the Senate of Universiti Pertanian Malaysia in fulfilment of the requirements for the degree of Master of Science.

INTRODUCING CONTROL AND STRUCTURE IN SOFTWARE PROTOTYPING

By

MD. MAHBUBUR RAHIM

March, 1992

Chairman : Dr. Abu Talib Othman

Faculty : Science and Environmental Studies

Software prototyping is emerging as an attractive software development paradigm in which a series of executable prototypes are constructed and users are encouraged to exercise with such prototypes in a live environment in order to solicit their overall requirements. In spite of these benefits, prototyping is not free from pitfalls. A major problem of software prototyping is the lack of explicit guidelines to control prototype iterations which tend to continue infinitely in a volatile environment. The problem is further aggravated by the unavailability of a suitable framework, within which to develop prototype systems in a manageable and flexible manner. Therefore, current practice of prototyping lacks in discipline.

This study is directed to address these critical issues of prototyping. The primary goal is to develop a strategy to control and to suggest a framework to manage software prototyping. A scheme called 'User satisfaction Method' which relates the degree of user satisfaction with the prototype's capability in clarifying user requirements is developed that provides rationale guidelines in deciding when to cease prototype iterations. To complement this scheme, a framework for structured prototyping, which is called 'State-Structured Transition' model is also developed. The framework considers each prototype 'version' as a 'state'and suggests that the transitions from one state to another need to be performed using structured principles.

In order to verify the applicability of such a framework and scheme, a case study has been undertaken. The results obtained confirm that 'User Satisfaction Scheme' can be adopted as a surrogate to control prototyping process. The research findings further establish that the framework of structured prototyping ensures smooth transition from one prototype version to another. Therefore, the 'User Satisfaction Scheme' should be adopted in conjunction with the framework of 'Structured Prototyping' in order to successfully control and manage software prototyping.

Abstrak tesis yang dikemukakan kepada Senat Universiti Pertanian Malaysia bagi memenuhi syarat untuk Ijazah Master Sains.

MEMPERKENALKAN KAWALAN DAN STRUKTUR DALAM PEMPROTOTAIPAN PERISIAN

Oleh

MD. MAHBUBUR RAHIM

Mac, 1992

Pengerusi : Dr. Abu Talib Othman

Fakulti : Fakulti Sains dan Pengajian Alam Sekitar

Pemprototaipan perisian sedang muncul sebagai satu paradigma pembangunan perisian yang menarik di mana satu siri prototaip yang boleh dilaksanakan, dibina dan pengguna digalakkan mencuba prototaip tersebut di persekitaran sebenar untuk memperolehi keperluan keseluruhan. Di sebalik faedah-faedah ini, prototaip juga tidak terlepas dari kelemahannya. Satu masalah besar pemprototaipan perisian ialah kekurangan petunjuk yang sesuai untuk mengawal lelaran prototaip dalam persekitaran yang berubah-ubah. Masalah ini menjadi bertambah rumit apabila tidak ada rangka kerja yang sesuai untuk membina sistem-sistem prototaip dalam keadaan terurus dan teratur. Oleh itu, amalan prototaip sekarang kekurangan dari segi disiplin.

Kajian ini ditumpukan kepada isu-isu kritikal dalam pemprototipan. Tujuan utama ialah membina satu strategi untuk mengawal dan mencadangkan satu rangka kerja untuk mengurus pemprototaipan perisian. Satu skema yang dipanggil 'Kaedah Kepuasan Pengguna' yang menghubungkait darjah kepuasan pengguna dengan keupayaan prototaip yang menerangkan kehendak pengguna telah dibina untuk memberi garis panduan yang rasional dalam membuat pertimbangan bila lelaran prototaip patut diberhentikan. Untuk melengkapkan skema ini, satu rangka kerja model prototaip berstruktur yang dinamakan 'Peralihan Kaedah Berstruktur' juga telah dibina. Rangka kerja ini mengambilkira setiap versi prototaip sebagai satu keadaan ke keadaan yang lain perlu dilaksana menggunakan prinsip-prinsip berstruktur.

Dalam mentahkikkan pengguna rangka kerja dan skema ini, satu kajian kes telah dibuat. Hasil daripada kajian ini telah mengesahkan bahawa 'Skema Kepuasan Pengguna' boleh diguna untuk mengawal proses pemprototaipan. Kajian ini juga telah berjaya membuktikan bahawa rangka kerja pemprototaipan berstruktur boleh mempastikan peralihan yang lancar dari satu versi prototaip ke versi yang lain. Oleh itu 'Skema Kepuasan Pengguna' patut digunakan selari dengan rangka kerja 'Pemprototaipan Berstruktur' supaya dapat mengawal dan mengurus pemprototaipan perisian dengan jayanya.

CHAPTER I

INTRODUCTION

Software prototyping as an alternative approach for developing Information Systems (IS), has recently been widely publicised in the computer literature. Also, there has been an increase in awareness among software community about the strengths of this new concept (Mayhew, Worsley and Dearnley, 1989). Recent survey studies (Guimares, 1987; Mohd. Hasan Selamat, 1988; Carey and Currey, 1989; Doke, 1990) confirm that prototyping approach is slowly gaining popularity. However, in spite of widespread publicity and discussions, still a large part of software developers remain sceptical and relatively few organisations have actually adopted this approach in developing their systems (Mayhew *et al.*, 1989). It is largely due to the lack of adequate know-how for developing, controlling and managing the evolving nature of prototyping. In this thesis, attempts have been made to examine the problems associated with the development and control of prototyping.

This chapter introduces the background of the problems associated with the systems developed by the traditional methods and presents prototyping as a possible solution to overcome these problems. However, in spite of enormous potential benefits, prototyping is not without any pitfalls.

This chapter also highlights the inherent weaknesses of prototyping that impedes its applications and argues for a need of research in order to overcome the drawbacks of prototyping. This chapter also presents objectives of the study to address the critical issues of prototyping and attempts to formulate new strategies to overcome these pitfalls.

Background of the Problem

During the last three decades, there has been a phenomenal proliferation of computers. This is partly due to the cost/performance ratio of the hardware which has improved drastically and to the micro-computer revolution. As computer has become more accessible, the demand for information in order to increase business opportunities and better management practices has also grown up (Yeh, 1990). This growing demand for computer solutions, coupled with increasingly complex and dynamic business environment, has resulted in the development of complex computer based information systems (CBIS). These systems are no longer as simple and small as they were in the early 60's. The information requirements of these systems have become more sophisticated, highly volatile and complex due to the competitive and dynamic business environment. Until now, most of these information systems are developed based on the classical waterfall model (Boehm, 1981; Fox, 1982; Sommerville, 1982), commonly known as Systems Development Life

Cycle (SDLC). Although the SDLC has been enhanced by adopting new techniques, its major weaknesses have not been eliminated (Slusky, 1987). In SDLC, attempting to elicit exact requirements is an extremely difficult task because of the cultural gulf that exists between systems developers and users. The second reason is that in SDLC, requirements are usually expressed in terms of natural language. Although natural languages are excellent medium for novels and poems, where the quality of such works is partly judged by the degree of ambiguity in the text, but their use in requirements specification may often lead to a disaster (Ince, 1988). Such textual descriptions are not suitable in communicating technical concepts between users and developers (Morrison, 1988). The third reason is that users do not consider system definitions to be a front end activity with a definite ending point, rather it is viewed as an ongoing process (Scharer, 1981).

An inadequate and inconsistent requirements are bound to introduce errors and such errors may sometimes be disastrous. Existing literature reports that many systems have failed due to the errors arising from misinterpreting requirements. Consequently, user requirements need to be identified carefully because good requirements is an important factor in the success of a system (Scharer, 1981). The SDLC provides little opportunities to overcome requirements identification problem. Many of the systems developed by SDLC result in unsatisfactory systems and frustrate end users. These systems are expensive to repair because the later the errors are detected, the more costly they are to repair (Davis *et al.*, 1989). Figure 1 highlights the fact that the cost of fixing an error rises dramatically as the software progresses through its life-cycle. However, sometimes, it is possible to salvage these systems, but only at the expense of high maintenance costs and frequently such maintenance costs far exceeds the development costs. Ince (1988) states that a survey conducted in U.S.A. finds that software professionals spend 48% of their time in maintenance while 43% time in development of new systems. Vonk (1990) mentions that total maintenance costs of

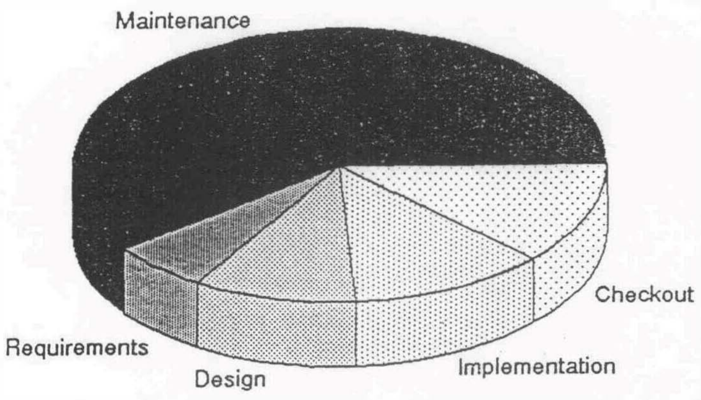

Figure 1 : Software Life Cycle : Per Error Fix Cost Per Phase

(After, Glass, 1981)

a system consumes 60% of the total life-cycle costs and 30% to 50% of the total life-cycle costs is due to inadequate analysis and understanding of user requirements. The net effect is that users are experiencing a growing degree of frustration with the seeming inability of the information systems professionals to meet their needs. These problems have created a tremendous bottleneck in SDLC. Consequently, the appropriateness of this model has been criticised by many researchers (Hekmatpur and Ince, 1988; Doke, 1990; Yeh, 1990) because of its inability to meet user requirements. As such researchers like Naumann and Jenkins (1982) suggest that new tools and methodologies should be ushered in to develop the right software within the constraints of budget, resources and schedule. In this study, an alternative software development paradigm known as software prototyping, has been addressed which has the potential to overcome many of the weaknesses associated with the SDLC approach.

The Significance of the Study

This study focuses on the software prototyping approach which is receiving growing attention and recognition from many academicians and practitioners - as a possible alternative solution to develop a right system based upon a clear and unambiguous requirements (Mahmood, 1987). In prototyping approach, a series of working models commonly known as prototypes are developed, to provide the users with an opportunity to interact with a real system to test their ideas and assumptions about the new system. Thus prototyping allows more concrete identification and validation of users information requirements (Alavi, 1984a; Naumann and Jenkins, 1982), facilitates systems implementation and acceptance (Appleton, 1983; Dearnley and Mayhew, 1983), improves communication between users and systems designers (Alavi, 1984a; Dearnley and Mayhew, 1983), significantly reduces maintenance efforts and shortens the over-all total life-cycle budget and schedule. Once the requirements are clearly understood and the proposed system is in development in the target environment, the prototypes may not be absolutely discarded, rather they can be wisely utilised as a training vehicle for the novice users (Morrison, 1988). These novice users can receive "hands-on" experience using the prototypes, well ahead of the proposed system is implemented. Prototyping enables the system to be designed from the users' perspectives (Harrison, 1985). Because of these benefits, the prototyping approach is often viewed as an 'insurance policy for success' in systems development (Guimares, 1987).

Prototyping is relatively a new concept in context to information systems development and like any new technology, it is not without any pitfalls. One major problem is that there exists little consensus among IS community on the definition, scope, environment, tools and benefits of prototyping. It is due to the fact that many authors have defined prototyping from a wide range of perspectives based on their own perceptions. Consequently, prototyping is frequently misinterpreted and misunderstood by the IS professionals and very often it is not properly practised. This ill-practice based on incorrect view of prototyping actually deprives IS people from the real benefits of prototyping. As such, a comprehensive definition needs to be proposed for software prototyping.

Another problem is that prototyping is often equated with the unstructured way of developing systems that were in practice before the arrival of structured methods and techniques (Vonk, 1990). It is primarily due to the lack of a discipline in developing systems using prototyping approach. Boehm and Standise (1983) mention that prototyping lacks a coherent methodology. In prototyping, the focus is on the identification of user requirements through a series of quickly produced prototypes. As such, developers have the tendency to rush to coding immediately after requirements are clarified through user evaluation of prototypes. Developers are tempted to plunge into prototype construction before sufficient analysis is carried out (Carey, 1990). Experiments conducted by Boehm, Gray and Seewaldt (1984) also confirm that in prototyping approach much less effort is devoted for planning and design. This clearly bypasses the analysis and design phases of prototype construction and consequently results in