CHAPTER 1: INTRODUCTION

In today's rapidly changing business environment, computerized systems are being widely used throughout business organizations to speed up business operations. While hardware components are essential requirements of a computer system, without the software components, a computer cannot be used to solve specific business problems. However, currently, there are many problems that occur during the development of software systems that lead to high development and maintenance costs.

1.1 Software Problems

Lientz and Swanson (1980) conducted a study of the amount of software maintenance effort required in 487 organizations. Lientz and Swanson's study showed that about 32% of the maintenance effort required were spent of fixing errors introduced during the development process.

In 1990 the Research Center on Software Engineering in Italy conducted a survey of maintenance activities in 35 companies (Mancini 1992). From this survey, COBOL was found to be the main language used by these companies and took up 70% of the language distribution. The survey also revealed that 25 of the 35 companies required more than 25% maintenance effort.

Tamai and Torimitsu (1992) conducted a survey of 42 Japanese organizations in 1991. The survey was designed to study the software replacement cases over a period of five years of each organization. They found that 60% of the software required reconstruction and 70% of the systems were written using COBOL. These results and those collected by the Research Center on Software Engineering in Italy confirm that Lientz and Swanson's figures are still applicable in the 90's.
Maintenance costs are high and cannot be ignored. It is generally agreed that maintenance alone absorbs approximately 50% to 80% of the software costs (Harrison 1990; Agha, G; Bull 1990; Buckley 1989; Parikh 1987; Simons 1987). Programmers also spend 50% to 80% of their time on maintenance (Harrison 1990; Parikh 1988; Kolodziej 1986). According to Swanson and Beath (1989), the maintenance proportion of the whole software cost will keep rising continuously due to the increases in software production by software developers.

Some of these software problems include: the inability of the conventional methodologies to handle the complexity of the systems being built; the inability of the conventional methodologies to cope with the rapid changes in the changing business environment; the lack of supporting tools to automate the development process. As well, there is a significant lack of uniformity in the notations used in the structured methodologies. English and different graphical notations are used during different stages of the development process (Ince 1988; Fichman and Kemerer 1992; Sutcliffe 1991; Weinberg, Guimaraes and Heath 1990).

Even though structured system development methodologies help in increasing productivity and reliability, such methodologies may cause software development problems (Sneed 1989). Even though it is useful for documenting a finished design, a top-down decomposition is not a good model for the design process. The problems which occur during the development process using the structured system development methodologies normally result in software being delivered late, over budget, and of poor quality (Bull 1990; Bielak 1993).
1.2 Software Engineering Goals
Poor quality software has led to high maintenance costs; therefore, software engineers are seeking ways of improving software quality to reduce maintenance costs. Besides looking for ways of obtaining quality software, software engineers are also searching for solutions to reduce development time and development costs. The three basic goals of software engineering are: to produce high quality software, to produce it at a lower cost, and deliver it on time (Mynatt 1990; Rine and Bhargava 1992).

1.3 Motivation for this project
Currently, Object-Oriented technology is a popular topic in the computing world. It has the potential to improve software quality and productivity (Burch 1993; Taylor 1990; Weinberg, Guimaraes and Heath 1990; Sharble and Cohen 1993; Garceau, Jancura and Kneiss 1993). Computer-Aided Software Engineering (CASE) is another technology that reduces development time and costs (Dixon 1992; Cheng and Ng 1990; McClure 1989; Fisher 1988). According to Martin (1993), Object-Oriented and CASE technologies fit well together, and the full potential of Object-Oriented methodologies will only be realized when the two technologies are combined. Combining these two technologies, all the three goals of software engineering (i.e., high quality software, shorter development time, lower development and maintenance cost) may be achieved. However, there is a shortage of Object-oriented CASE tools at present and there is no universal standard for Object-Oriented methodologies (Bielak 1993; Henderson-Sellers 1992a; Ward 1990; Taylor & Hecht 1990; Loy 1990; Weinberg, Guimaraes and Heath 1990).

System designers are becoming more and more interested in object technologies, not only because of the recent explosion of methods and tools and the hype about them, but also because of the technological revolution they seem to imply. These potential users see in them a paradigm closer to the real world, an abstraction capability higher than traditional approaches, and above all, the long dreamed of benefits of reusing components and of a modularity allowing an easy evolution of system.

But what are the real consequences, advantages and drawbacks of those technologies on a large-scale system, which is equivalent to hundreds of thousands of traditional lines of code? What
about those organizations that have systems developed using traditional systems development life cycle that are working perfectly now and wish to incorporate OO technology in these systems. How do we reengineer legacy systems? Will this be impossible with the constraints of budget, time, and market share? This project proposes a methodology that could be used to address these issues.

1.4 Objectives of this project
The objective of this project is to review the strengths and weaknesses of both structured and object-oriented methodologies and attempts to propose a new life-cycle paradigm that addresses these issues. This life-cycle model, referred to as the hybrid convergence analysis model (HYCONAN), filters the weaknesses of both methodologies and incorporates their strengths. The model derives its strength from a number of features that most software engineers would agree as necessary prerequisites for successful systems development. These features are: faster software production, reengineering of legacy systems, reusability of components, concurrent system development, consistent user involvement in the development of systems, and a continuous process of refinement through validation and verification up to the point of systems implementation. In order to achieve the required model, a review of the matter is presented. This comparative review may be used to justify the real needs for the development of a software development process model.
1.5 Thesis Organization

Chapter 1: Presents an overview of the thesis including the goals, motivation and objectives of the thesis.

Chapter 2: A literature review of structured and object-oriented paradigms. An in-depth comparative review is done, emphasizing on the strengths, weaknesses and the techniques used.

Chapter 3: Discusses and justifies the Hybrid Convergence Analysis model, including its derivation.

Chapter 4: Presents an in-depth analysis of the concepts and techniques used in the hybrid model.

Chapter 5: Discusses CASE tool in general together with its benefits and the development of the HYCONAN CASE tool using MetaEdit.

Chapter 6: Concluding remarks on the contribution of the project, limitations and the direction required for future enhancements to the project.