

# Preface

The development of software systems is now regarded as among the most complex tasks performed by mankind. The problems that are caused by the scale of this complexity have been recognized for a long time. This complexity affects the costs and time expended on software creation. Moreover, after being built, software systems are often unreliable, difficult to use and, what is worse, they are frequently difficult to maintain and evolve. These difficulties, together with the ever-increasing demand for software, have led to what has become known as the *software crisis*.

An idea which has been receiving a great deal of attention from software engineers is the *object-oriented paradigm*. Currently, this paradigm is thought to be an important aspect of software development, so much so that it has become a major area which is expected to bring significant benefits to software production. The rapid evolution of this paradigm can be attributed to several important reasons, which are discussed in more detail later in this book, but which include: better modelling of real-world applications; better structure for software systems based on abstract data type concepts; and the possibility of reusing software during the creation process.

Despite all of the progress so far in the object-oriented arena, there is a gap in the knowledge concerning software construction. That is, despite the acknowledged importance of software development methodologies and the popularity of object-orientation, there is no generally accepted software methodology that essentially addresses object-oriented development and considers reusability as part of a software life cycle model. From the theoretical and practical viewpoints, the application of object-oriented methodologies remains a topic of major interest.

On the other hand, most books on software engineering has focused mainly on the development phases of the software life cycle; that is, analysis, design and implementation. Until recently, however, software maintenance has been the neglected phase in the software engineering process. The literature on maintenance contains very few entries when compared to the development phases. Little attention has been placed into the subject, and consequently few software maintenance methodologies have been used.

The maintenance of legacy systems (existing software systems) may account for over fifty per cent of all efforts expended by an organization that deals with software. The percentage continues to rise as more software systems are produced. Additionally, as systems age, more effort is likely to be expended on maintenance. Although most legacy systems should be retired or periodically rewritten, this is not always feasible. Since most organizations are gradually becoming more dependent on legacy systems, software maintenance is crucial.

Notwithstanding increasing recognition that maintenance is a major problem during the life cycle of software systems, and acknowledgment of the high costs of maintaining software, there are clearly gaps in the field of software maintenance. So far, there is no generally accepted methodology that systematizes the software maintenance process; neither is there an integrated set of tools which helps tackle the problem of controlling software changes under a methodical scheme.

This book aims at presenting a generic methodology to teach object-oriented design, which considers software reusability as an im-

portant aspect of the software life cycle. The main features obtained through the use of an object-oriented design methodology is the creation of a software system following strictly an object-oriented approach, as such a methodology concentrates on identifying and representing classes, inheritance and objects. In doing so, the architecture of an object-oriented software at the design level is built around sets of classes and objects. Moreover, considering reusability as a pragmatic (and desirable!) process within the design phase helps the designer relate software components to each other through relationships which show where a component is defined and used, and in what context. In this way, reusability is encouraged within a software life cycle as part of the object-oriented design methodology.

The book is equally concerned with the description of a general methodology to teach software maintenance. This methodology provides guidelines and procedures for carrying out the maintenance process, while establishing a systematic approach for the support of legacy systems. The methodology takes the software configuration management discipline into account, since this discipline imposes a set of procedures and standards for managing evolving software.

Additionally, there have been a growing interest not only in new methodologies for software development and maintenance, but also the way in which these methodologies can be supported by computerized tools. This technology, known as CASE (Computer-Aided Software Engineering), allows software engineers to document and model a software system from its initial requirements through the design and maintenance stages. Consequently, the ultimate objective for any methodology is that it should be automated by CASE tools. By this means, the rules, principles, guidelines and graphical notations set up by the methodology can be enforced and followed by software engineers, and inconsistencies can be exposed. Ideally, the tools should be integrated into a CASE environment through a common interface together with a single database used to store software information, such as names of classes, attributes, operations and objects, in a uniform representational model.

## Aims of the Book

As a book that tackles two important aspects of the software life cycle, i.e. design and maintenance, it gives emphasis to a life cycle model which explicitly recognizes the importance of reusability during the design and maintenance phases. The primary goals of this book are:

- to classify existing object-oriented methodologies according to their suitability for a particular phase of the software life cycle and their application domains. This classification can be used to understand which methodology is best applied to specific phases of the life cycle or certain kinds of systems;
- to present a methodology to teach object-oriented design independent of any other methodology or any programming language. This methodology can be applied to designing software systems which will then conform to object-oriented concepts such as classes, objects and inheritance;
- to offer an alternative software life cycle model within an object-oriented framework, which emphasizes the importance of software reuse during the development and evolution of software;
- to describe a methodology which provides guidelines and procedures for the maintenance process by applying the software configuration management discipline. This methodology improves the balance between source code and higher level abstractions after any change in the software system has been made. As a result, a reliable and easily understandable documentation of an existing software system can be incrementally obtained while it is being maintained;
- to define a software maintenance model, whose phases institute a change control framework to monitor changes. Such a model aims at systematizing the software maintenance process by specifying the chain of events and the order of stages that a change has to go through.

## Outline of the Book

The book is organized into eight chapters as follows.

Chapter 1 expands upon the background of the object-oriented paradigm and is divided into six sections. The first section introduces the notions of application domains and solution domains, and shows how object-oriented thinking helps bridge the gap between these two domains. Section 2 discusses the profile of present software in terms of complexity, friendliness, extensibility and reusability. The road towards an object-oriented approach is explained in Section 3, which also presents an overview of the most familiar object-oriented programming languages. Section 4 reviews the main concepts related to object-orientation, and introduces the terminology associated with the object-oriented paradigm to be used in the remainder of this book. The terminology to be presented is independent of any programming language or system. The fifth section introduces the philosophy upon which object-oriented design is based, and examines important issues associated with software construction such as domain analysis, reusability and software life cycle models. A summary of this chapter is given in the sixth section.

Chapter 2 considers a series of classification schemes to assist in the understanding of several existing object-oriented methodologies. The first section compares and contrasts the differences between structured development and object-orientation. The second section introduces the most well-known methodologies, methods and techniques to tackle the analysis and design phases of object-oriented software development. This section discusses the different terminologies employed by them and evaluates their advantages and limitations. Final remarks on this survey are presented in the third section.

Chapter 3 contains a detailed description of a methodology for object-oriented design based on the concepts described in the first chapter. The first section places the methodology into the context of software development. The steps which must be followed in order to design an object-oriented software are discussed in the second section. A means of graphically representing the produced design with a series

of different diagrams is explained too. Section 3 gives feedback based on experience with the methodology. The requirements for a CASE environment are examined in the fourth section. The chapter concludes with a review of the methodology steps, outlining issues which must be considered when designing software.

Chapter 4 provides an account of reusability and life cycle issues which arise during object-oriented software production. The first section outlines existing mechanisms to achieve reusability. Besides, it concentrates on the main reasons why software components are not frequently reused and discusses the complications associated with reusability during the design phase. Section 2 puts forward a new software life cycle model that addresses reusability within an object-oriented framework. The section also considers the role of the knowledge about the application domain, and discusses how the knowledge that the designer has about the application domain can affect the creation of software in terms of a top-down, bottom-up or middle-out manner. Some important aspects related to the fully applicability of the presented methodology for object-oriented design within the planned software life cycle model are examined throughout section 3. The chapter finishes with comments on the software process presented in the first two sections.

Chapter 5 expands on the general background of software maintenance and the necessary documentation, as well as introduces the software configuration management discipline to maintain legacy systems. The first section introduces the concepts, problems and categories of software maintenance. Section 2 concentrates on the supporting maintenance technology. In Section 3, the documentation necessary to maintain legacy systems and an approach for incremental documentation are presented. Section 4 looks at the software configuration management discipline, which has been applied mainly to the software development process, as another important factor in helping the maintenance of legacy systems. A summary of this chapter can be found in Section 5.

Chapter 6 describes existing software maintenance models. Section 1 examines the use of models to manage the software life cycle with maintenance. In the second section, a series of software maintenance models, aimed solely at maintaining legacy systems, is presented and compared. The third section introduces the notions and desirable characteristics of software process modelling for software development and maintenance. Section 4 identifies specific points of the software maintenance modelling considered in this book. A summary of this chapter is presented in the fifth section.

Chapter 7 presents a detailed description of a methodology for the maintenance of legacy systems. Section 1 examines the main objectives of such a methodology. In Section 2, a software maintenance model together with its phases is detailed. The role of the software configuration management discipline and the application of each configuration function to the software maintenance methodology are discussed in Section 3. Section 4 details the version control applied to the outcome of the methodology's phases. The chapter concludes with observations on the described methodology.

Finally, Chapter 8 concludes this book by reiterating its main points. Section 1 combines the object-oriented paradigm along with software maintenance. The second section summarizes the methodologies for object-oriented design and maintenance of software systems. The third section provides final remarks on the application of object-orientation. The last section presents an overall conclusion regarding the future of object-oriented software engineering.

## Acknowledgments

It is our pleasure to acknowledge some people who have influenced the writing of this book.

We would like to thank Professor Peter A. Lee (University of Newcastle upon Tyne, England) and Professor Malcolm Munro (University of Durham, England) for their comments and constructive criticisms on early drafts of the methodologies put forward in this textbook. We are also grateful to Professor Toshiyasu L. Kunii (University of Aizu, Japan) for allowing us the necessary time to complete this project.

Special thanks are due to our families, in Brazil, for understanding our absence during the moments they most needed us. Their endurance has been really admirable. Our parents, especially, have given us strength and motivation throughout our careers, and for teaching us the meaning of perseverance. Their efforts are greatly appreciated and will never be forgotten.

*Luiz Fernando Capretz*  
*Miriam Akemi Manabe Capretz*

University of Aizu, Japan  
March, 1996