# Preface

## About This Book

Most requirements books today provide general-purpose guidance such as "involve the customer" and "make the requirements testable," or document-specific techniques such as Use Cases. In spite of this breadth of coverage, several important topics are weakly, rarely, or never covered in requirements books. These topics include the effect of requirements on overall software quality (weakly covered), requirements reuse (rarely covered), and requirements encapsulation (never covered). As its title suggests, *Software Requirements: Encapsulation, Quality, and Reuse* strives to remedy these shortcomings.

This book is able to cover these additional topics because it focuses on the concepts and techniques of the Freedom approach to requirements. Freedom is a lightweight, customer-centric technical software development methodology originally developed for NASA's Space Station Freedom Program. Freedom strives to meet customer needs for functionality by specifying requirements in an innovative manner that permits *encapsulation* of requirements in code objects for later ease of change. Requirements encapsulation in turn enables requirements *reuse*. Customer needs for quality are addressed by continuous emphasis on quality drivers throughout the development process. Direct participation of the customer, or a knowledgeable customer representative, is essential to the Freedom requirements process.

Freedom's approach to requirements involves a change in perspective. Rather than viewing requirements as statements about the software, Freedom considers requirements to be part of the software, namely, its external interface. Freedom involves customers in requirements specification by enlisting their help to specify the external interface of the software that they will use. With the assistance of the developers, customers specify

the software external interface in terms of stimulus–response pairs organized into cohesive sets called "stimulus sets." The stimulus sets are themselves organized hierarchically into a "functionality tree" that defines the architecture of the external interface. During design, developers use the external interface architecture as the upper level of the design architecture, thus ensuring architectural identity between requirements and design. Within this upper level of design, a requirements encapsulating "functionality module" is created for each stimulus set of the functionality tree, thus ensuring architectural identity between requirements and implementation. A change to any requirement (external interface stimulus–response pair) is consequently localized by the architecture to one functionality module. Architectural symmetry effectively achieves requirements encapsulation in code modules, making requirements change easier throughout the life of the software, and enables requirements reuse, easing future development.

It is suggested that readers of this book have some prior understanding of object-oriented (OO) concepts. An OO background is helpful in understanding the Freedom concept of requirements encapsulation, which is built upon the OO concept of information-hiding. Due to its importance, information-hiding is reviewed in Chapter 2, but prior exposure to OO can ease comprehension.

Coding proficiency is also helpful in getting the most out of this book. Coding may seem like an unnecessary prerequisite for a requirements process. However, creation of a user interface (UI) mockup is a necessary step in the process. A UI mockup is program code that implements the proposed look and feel of the UI. It is an effective vehicle for obtaining user confirmation of requirements correctness very early in the development cycle when change is least expensive. Also, an explanation of the structure of a UI mockup provides insight into the practical aspects of encapsulation of requirements in code objects. For these reasons, the book covers creation of UI mockups. Hence, prior exposure to coding in general, and UI development in particular, is helpful.

The preferred programming language for Freedom is an OO language such as Java. Hence, the code examples in the book are in Java. However, Freedom can be used with any programming language that supports data encapsulation. This includes non-OO languages such as C or Fortran when such languages are used carefully.[1]

This book uses terminology from original information-hiding theory, and from modern object-oriented languages such as Java. Both sources use different words to describe the same or similar concepts. For example, the terms "module" and "class" both refer to a unit of code. Generally speaking, "module" is a generic term for a unit of code, and "class" is a unit of code in an OO programming language such as Java. Clarification of such terminology is provided by the Glossary.