Preface

In order to establish itself as a branch of engineering, a profession must understand its accumulated knowledge. In addition, software engineering as a branch of engineering must take several basic steps in order to become an established profession, highlighting understanding of the nature of its knowledge.

Software engineering experts always have used proven ideas. Concretely, in the object-oriented (OO) design knowledge field, the practical experience of it has been crucial to software engineers, and it is in the last years when these ideas, materialized in items such as patterns or refactorings have reached their biggest popularity and diffusion. And in this regard, the software engineering community has advanced greatly and we currently have numerous and defined chunks of knowledge, including standards, methodologies, methods, metrics, techniques, languages, patterns, knowledge related to processes, concepts, and so forth. Although these different areas of knowledge relate to the construction of an OO system, there is a lot of work still to be done in order to systematize and offer this knowledge to designers in such a way that it can be easily used in practical cases.

A software architecture is a description of the subsystems and components of a software system and relationships between then.[1] Usually, the software architecture is subdivided into macro and micro architecture. Whereas macro architecture describes the metamodel of design, this that provides the high-level organization, the micro architecture describes details of a design at a lower level.

OO design is a software design technique, which is expressed in terms of objects and relationships between those; at the level of micro architecture it includes elements such as classes, its relationships, responsibilities, refactorings, and so on.

OO micro architectural knowledge is built upon design experiences, such as problem solving, or lessons learned. Therefore, the OO micro architectural design knowledge has grown with time and the increasing complexity of software. This knowledge expands and accumulates when it is stored in books and other media for the use of designers.

In addition, the major part of OO design knowledge is difficult to identify and use. The experience has demonstrated that design often omits common principles, heuristics, and so on, with a consequent major loss of experience. Consequently, actually, serious difficulties are still encountered when we tackle the construction of OO systems. Although designers have accumulated a body of knowledge that they apply during these processes, this is very implicit. Fortunately, it is now being specified and popularized in different forms: principles, heuristics, patterns, and more recently, refactoring techniques. However, today, the difference between these concepts is generally unclear and not all of them have received the same amount of attention or have reached the same degree of maturity. In addition, a strong knowledge does not exist on items such as design principles, best practices, or heuristics. The problem confronting the designer is how to articulate all this explicit knowledge and to apply it in an orderly and efficient way in the OODA, in such a way that it is really of use to him or her. In fact, in practice, even such advanced subjects like OO patterns have this problem

Design knowledge and best practices are stored in individual expert minds, or implicitly encoded and documented in local organisational processes. It has always been true that a significant part of design knowledge resides in the minds of the experts that make it up. However, communities and companies are beginning to find that it is easy to lose a vital element of their intellectual property: corporate design knowledge. Therefore, we can say that the major part of the design knowledge today is tacit knowledge: it in the form of project experiences, heuristics, or human competencies that are difficult to be captured and externalised.

The effective management of this knowledge is today a significant challenge. For knowledge management to be effective, this knowledge should be organized and classified. In addition, with this purpose, developing unified catalogues of knowledge, ontologies, empirical studies, and so on, books and studies such as those we present here, are very important issues to improve the use of OO design knowledge.

Therefore, in this context, we present this book whose main objective is to give a global vision of micro-architectural design knowledge, exposing the main techniques and methods, and analyzing several aspects related to it.

The subject matter in this book is divided into ten chapters. The chapters seek to provide a critical survey of the fundamental themes, problems, arguments, theories, and methodologies in the field of OO micro architectural design knowledge. Each chapter has been planned as a self-standing introduction to its subject.

Therefore, in **Chapter I** Javier Garzás and Mario Piattini present an introduction to "The Object-Oriented Design Knowledge," where they show the main issues and problems of the field. In OO micro-architectural design knowledge, design patterns are the most popular example of accumulated knowledge, but other elements of knowledge exist such as principles, heuristics, best practices, bad smells, refactorings, and so forth, which are not clearly differentiated; indeed, many are synonymous and others are just vague concepts.

An essential issue to building an OO design knowledge discipline is organizing this knowledge. In **Chapter II**, titled "The Object-Oriented Design Knowledge Ontology," Javier Garzás and Mario Piattini show an ontology that organize and relation the OO knowledge. The authors propose an ontology in order to structure and unify such knowledge. The ontology includes rules (principles, heuristic, bad smells, etc.), patterns, and refactorings. They divide the knowledge on rules, patterns, and refactorings and they show the implications among these. Moreover, they show an empirical validation of the proposed conclusions.

Chapter III, "Using Linguistic Patterns to Model Interactions," by Isabel Díaz, Oscar Pastor Lidia Moreno, and Alfredo Matteo, is a pivotal chapter that changes the focus of the book to more technical information systems issues. This chapter shows an elegant example of how highly relevant clinical questions can be addressed in a scientific manner. In this chapter, heuristic-oriented techniques and linguistics-oriented techniques proposed by several authors to model interactions are analyzed. In addition, a framework to facilitate and to improve the interaction modeling is described. This framework was conceived to be integrated into automatic software production environments. It uses linguistic patterns to recognize interactions from use case models. The validation process used and the main results are also presented.

In **Chapter IV**, Manoli Albert, Marta Ruiz, Javier Muñoz and Vicente Pelechano show "A Framework Based on Design Patterns: Implementing UML Association, Aggregation and Composition Relationships in the Context of Model-Driven Code Generation." The chapter proposes a framework based on design patterns to implement UML (Unified Modeling Language) association, aggregation, and composition relationships, and for it they propose a semantic interpretation of these concepts that avoids the ambiguities introduced by UML.

Therefore, in "Design Patterns as Laws of Quality" Yann-Gaël Guéhéneuc, Jean-Yves Guyomarc'h, Khashayar Khosravi, and Houari Sahraoui, **Chapter V**, show how design patterns can be used as facts to devise a quality model and they describe the processes of building and of applying such a quality model.

The chapter highlights the need for principles in software engineering, where these can be laws or theories formalizing and explaining observations realized on software.

For the sake of completeness in this book, automatic verification of design knowledge is addressed in **Chapter VI**. Andres Flores, Alejandra Cechich, and Rodrigo Ruiz present "Automatic Verification of OOD Pattern Applications."

Chapter VII, "From Bad Smells to Refactoring: Metrics Smoothing the Way", is authored by Yania Crespo, Carlos López, María Esperanza Manso Martínez, and Raúl Marticorena. This chapter discusses one of the current trends in refactorings: when and where we must refactor. From the bad smell concept, it is possible to discover their existence from an objective viewpoint, using metrics. The chapter presents a study on the relation of refactorings, bad smells and metrics, including a case study on the use of metrics in bad smells detection. The chapter leads to the determination where refactoring is the basis of heuristics and metrics, which is likely to be the single most important factor at the moment of use refactorings in the maintenance phase.

Therefore, in **Chapter VIII**, "Heuristics and Metrics for OO Refactoring: A Consolidation and Appraisal of Current Issues," Steve Counsell, Youssef Hassoun, and Deepak Advani cover this topic in great depth. They look at some of the issues which determine when to refactor (i.e., the heuristics of refactoring) and, from a metrics perspective, open issues with measuring the refactoring process. They thus point to emerging trends in the refactoring arena, some of the problems, controversies, and future challenges the refactoring community faces.

A key point to building a OO design knowledge field is to understand the several contributions to it. Since several OO metrics suites have been proposed to measure OO properties, such as encapsulation, cohesion, coupling, and abstraction, both in designs and in code, in **Chapter IX**, titled "A Survey of Object-Oriented Design Quality Improvement," Juan José Olmedilla reviews the literature to find out to which high level quality properties are mapped and if an OO design evaluation model has been formally proposed or even is possible. The chapter is an excellent example of how performing a systematic review of the estate of art.

At last, in **Chapter X**, "A Catalog of OOD Knowledge Rules for OO Micro-Architecture," by Javier Garzás and Mario Piattini, several types of knowledge such as principles, heuristics, bad smells, and so on, are unified in a rules catalog.

In summary, these chapters constitute an evidence of the importance of micro-architectural design knowledge, representing important ideas in different software design areas. These are intended to be useful to a wide audience, including software engineers, designers, project managers, software architects, IS/IT managers, CIOs, CTOs, consultants, and software students.

We hope that the practical vision, scientific evidence and experience presented in this book will enable the reader to use the design knowledge within the field of software engineering and to help the field of software engineering answer how software engineers might acquire its rich and essential accumulated knowledge.

Javier Garzás and Mario Piattini, Editors
Ciudad Real, Spain
January 2006

Endnote

[1] Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., & Stal, M. (1996). *A system of patterns: Pattern-oriented software architecture*. Addison-Wesley.