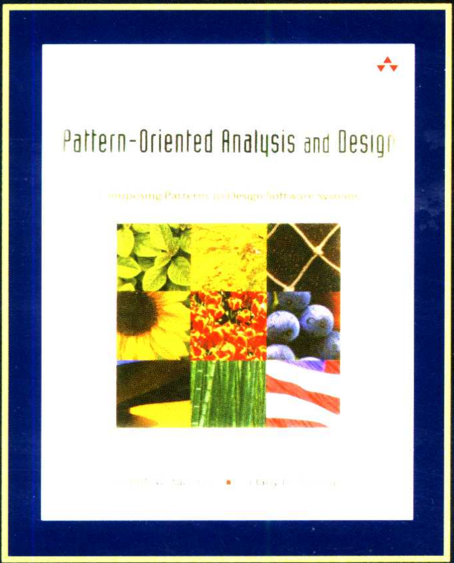
原 版 风 暴 系 列

PEARSON
Addison Wesley

Pattern-Oriented Analysis and Design
Composing Patterns to Design Software Systems

面向模式分析和设计
通过模式合成来进行软件系统的设计
（影印版）

[美] Sherif M. Yacoub
Hany H. Ammar 著



讲述 POAD 方法、原理、过程，包括 4 个大型范例 ■
使用 POAD 方法和 UML 类图可快速开发大型系统 ■
适合软件架构师、设计开发人员和项目经理阅读 ■

Hany H. Ammar

中国电力出版社
www.infopower.com.cn

原 版 风 暴 系 列

Pattern-Oriented Analysis and Design
Composing Patterns to Design Software Systems

面向模式分析和设计
通过模式合成来进行软件系统的设计

(影印版)

[美] Sherif M. Yacoub
Hany H. Ammar 著

中国电力出版社
www.infopower.com.cn

图书在版编目（CLP）数据
面向模式分析和设计：通过模式合成来进行软件系统的设计 / （美）亚克布（Yacoub, S. M.），（美）阿马尔（Ammar, H. H.）编著. —影印本. —北京：中国电力出版社，2004
（原版风暴系列）
ISBN 7-5083-2209-6
I. 通... II. ①亚... ②阿... III. 软件设计-英文 IV. TP311.5
中国版本图书馆 CIP 数据核字（2004）第 032344 号

从 书 名：原版风暴系列
书　　名：面向模式分析和设计：通过模式合成来进行软件系统的设计（影印版）
编　　著：（美）Sherif M. Yacoub, Hany H. Ammar
责任编辑：姚贵胜
出版发行：中国电力出版社
　　　　　地址：北京市三里河路 6 号　　　邮政编码：100044
　　　　　电话：（010）88515918　　　　传　　真：（010）88518619
印　　刷：北京丰源印刷厂
开　　本：787×1092　1/16　　　　　印　　张：24.75
书　　号：ISBN 7-5083-2209-6
版　　次：2004 年 5 月北京第 1 版　　　2004 年 5 月第 1 次印刷
定　　价：39.80 元
版权所有　翻印必究

To my mother Zienab Halawa, and the memory of my father Mohamed Yacoub
To my sister Noha Yacoub and my brother Yasser Yacoub
To my uncle Samir Halawa
To my professors Hany Ammar and Ali Mili

Sherif Yacoub

To my dear mother Lila Awny, and the memory of my late father Hussein Ammar,
To my wife Mona, my children Kareem, Taher, Hussein, Mariam, and Hoda,
To my sisters Azza and Hala, and to my aunts Ragia and Samiha
To the Engineering Faculty of the International Islamic
University Malaysia, where the background on
Design patterns caught my attention
During my leave at IIUM
In the spring of
1995

Hany Ammar

Preface

The most difficult part of building software is not coding; it is the decisions you make early at the design level. Those design decisions live with the system for the rest of its lifetime.

Although this statement might offend many software developers who are strong believers in coding and implementation, in reality it should not. It is more a compliment than an insult! How?

We believe that design is a critical phase of the software development lifecycle. Good design decisions eventually result in a good product, and bad design decisions generally affect the quality of the final product. But the question is, How do we make a good design decision, and how can we assess that such a decision is "good" when we do not have the final product to test our decisions early at the design stage?

Software engineering disciplines realized this paradox a long time ago. The shift from a waterfall development lifecycle to an iterative and rapid prototype lifecycle is a solid proof. In any organization, prototyping is a well-appreciated practice. When you do not know whether your idea is implementable, try to implement a reduced version and test it. When we do so, we learn things at the late implementation and coding stage that we were not aware of at the design stage.

Wouldn't it be great to have someone we trust to whom we can explain the problem and get some useful answers like, such as "I have implemented this before; it does not work because..." or "I have implemented it before, and this is the way to do it. Moreover, this implementation has these advantages and these drawbacks."

Design patterns were introduced to serve as the advice from the expert. The real power behind patterns is that they are abstractions from the real world. Experienced software designers and developers have implemented and tested solutions to recurring design problems. Design patterns capture their experiences and present them to all other designers in a form that defines what problem is being solved, how it is solved, why the solution is good, and the implications of using that solution.

Therefore, design patterns are captured by experienced software developers and designers during implementation of many applications, then documented at the design phase (possibly with some implementation examples). Other designers then use and deploy these patterns in developing new applications.

The design decisions that we make at the design phase are crucial to the application development. We need to make good design decisions. In reality, we cannot make good design decisions unless we have the experience that enables us to make these decisions. Experienced software developers and designers have this experience, and they convey it to us in the form of design patterns. Hence, software developers make a major contribution to the design process by documenting and presenting those patterns in a form that is usable at the design stage. They make life easier for designers by providing them with these bulletproof, good design solutions. Whereas we believe that design is a critical phase in software development, we also believe that experienced developers and designers have so much to contribute to make the design process a success.

WHAT DO WE REUSE?

Reusing software is one approach to expedite the software development process. The question is, then, What can we reuse and how? Code is the most common form of reuse. Before developing a software component, we actively browse the Internet for open source code that we can borrow, modify, and reuse. Reusing designs is a less frequent practice than reusing code due to the complexity and difficulty of constructing generic designs and instantiating them. Moreover, code is more tangible than design, since we can deploy and execute code with little or no modifications. However, it is very unlikely that we will find a black box component to satisfy all our requirements. It is also very risky to modify the source code (if it is available), since this may break the component integrity and the functionality for which it was originally built. Therefore, many software developers prefer to reuse the idea of the solution and have it implemented their way. Designs are presented at higher levels in the form of design models that require further instantiation and implementation. Design patterns help in leveraging the reuse level to the design phase by providing the design models (and sample implementations) that can be reused.

COMPOSING DESIGN PATTERNS

When we browse existing work and literature on design patterns, we realize that most of the effort is expended in discovering and documenting patterns, and little work is concerned with systematically applying these reusable designs in developing new applications. The problem that deserves more attention is how to compose design patterns to develop software and how these composition approaches are supported by versatile design models such as the Unified Modeling Language.

We generally classify approaches to design applications using patterns as follows:

1. *Incidental or ad hoc.* A design pattern provides a solution together with the forces and consequences of applying this solution. However, this is not usually sufficient to systematically develop with patterns. For instance, the coincidental use of a Strategy pattern in the implementation of a control application is not a systematic approach to deploy patterns. This is simply because there is no process to guide the development and to integrate the pattern with other design artifacts. Hence, the design process is not repeatable.

2. *Systematic.* A systematic approach to design with patterns goes beyond just applying a certain pattern. Systematic approaches can be classified as
   a. Pattern Languages. A pattern language provides a set of patterns that solve problems in a specific domain. Pattern languages not only provide the patterns themselves but also the relationships between these patterns. They imply the process to apply the language to completely solve a specific set of design problems.
   b. Development processes. A systematic development process defines a pattern composition approach, analysis and design steps, design models, and tools to automate the development steps.

We advocate a systematic development processes for developing with patterns, since this is the only way to make design patterns a common practice in software development. To improve the practice of *systematically* deploying design patterns in developing software, we need to

- Define composition techniques that can be used to construct applications by composing design patterns, and
- Support these composition techniques with appropriate modeling languages and views.

POAD

While a great deal of research and practice has been devoted to discovering new design patterns, very little has been concerned with the systematic process of "gluing" and "composing" design patterns to develop software applications. This book specifically addresses this problem and provides a practical methodology to compose and deploy design patterns.

This book presents an approach to design software applications using design patterns. It describes a POAD methodology that produces pattern-oriented designs. POAD takes a structural composition approach to glue patterns at the high-level design. It uses the notion of constructional design patterns as design components with interfaces.

POAD is based on the premise that at some design level, it is sufficient to know that some patterns are used in the application, and it is not necessary to overwhelm the designer with the details of the internal design of each pattern. Wouldn't it be nice to work at a higher level than class diagrams and yet know that elements at that level have well-proven class diagrams? This is achieved by POAD. POAD provides logical views to represent the application design as a composition of patterns and provides the necessary means to trace participants of those patterns into the application's final class diagram.

The book provides a briefing of existing design pattern composition approaches and then describes an example-driven methodology to develop robust software designs using patterns as their building blocks. The book describes the technological aspects and the process aspects of the methodology. The technological aspects focus on the models required to glue design patterns together, and the process aspects walk the designers and architects through the various analysis and design steps.

AUDIENCE

The intended audiences for this book include the following:

1. Software architects and software designers seeking illustrative techniques to deploy patterns in designing software applications. They will learn how to construct robust, maintainable software architectures using design patterns as their building blocks.
2. Practitioners seeking state of the art and practice in applying design patterns and learning about using pattern catalogs to build software.
3. Application developers seeking benefits from applying design patterns early at the design level rather than applying them only at the code level. The case studies described in the book give hands-on experience in applying design patterns. The examples in Part IV provide useful illustrations.
4. Computer science and software engineering students learning about using design patterns as a good software engineering practice in designing applications. The case studies help students understand and apply basic design patterns to develop applications.
5. Researchers seeking state of the art in pattern composition and an understanding of related issues that could be topics of future research initiatives. The book helps researchers unveil the research problems in design pattern composition.
6. The book helps professors and lecturers in preparing design patterns courses, case studies, and projects by serving as a reference book for pattern composition approaches as well as simple and complex case studies.
7. Reuse managers who want to learn how reusing design patterns can be useful in building a robust, maintainable software architecture. The book helps organization managers in adopting design pattern reuse programs by illustrating an easy-to-apply methodology that can be used early in the software development process.

Prior Knowledge

The following is the background required for the audience:

- Knowledge of the Unified Modeling Language, specifically class and package diagram models.
- Basic OO design concepts, including inheritance, delegation, aggregation, and so on.
- Knowledge of the basic concepts of design patterns, including what patterns are and familiarity with some pattern examples from any pattern catalog book.

Scope

This book is *not* about

- Teaching OO design models such as the UML.
- Teaching the basics of design patterns.

- Documenting new design patterns.

  This book *is* about

- Deploying design patterns in software development.
- Composing design patterns.

How to Read the Book

Part I introduces POAD concepts. Chapter 1 is an introduction to the POAD methodology. In Chapter 1 we discuss the type of problems that POAD solves. Chapter 2 discusses the role of patterns in software design. In Chapter 3 we classify design pattern composition approaches into structural and behavioral composition mechanisms. Chapter 3 elaborates on the behavioral and structural composition approaches respectively and discusses examples from existing design composition techniques. The chapter contains references to further readings on these composition mechanisms.

Part II discusses the technological aspects of POAD. In Chapter 4 we discuss the role of design patterns as building blocks of software design and which design patterns can be used with POAD. Chapter 5 introduces the design models that we use to compose design patterns. It also shows how the UML syntactically supports these models. In Chapter 6 we discuss the UML support for design patterns. We compare different UML approaches to model design patterns and their composition.

Part III discusses the process aspects of POAD. Chapter 7 describes the procedures to apply POAD in the design of software systems. We first discuss the stringing and overlapping pattern composition approaches, then illustrate how POAD reaps the benefits of the two worlds. This chapter also summarizes the analysis, design, and design-refinement phases of POAD and illustrates the overall outline of the process. Chapters 8, 9, and 10 elaborate more on the analysis, design, and design-refinement phases respectively and discuss the development and modeling steps within each phase.

Part IV provides case studies and illustrates the application of the POAD methodology to develop pattern-oriented designs and frameworks. We show examples of applying the methodology to four case studies.

Chapter 11 illustrates the application of POAD in the development of a pattern-oriented design framework for feedback control systems as an example of reactive systems. The framework is generic and is easily instantiable in developing application-specific control systems.

Chapter 12 illustrates the application of POAD in the development of a pattern-oriented design for the domain of simulation of waiting queues as an example of a product line. This framework deals with simulation of customers lining up for service from one or more service stations, such as the supermarket checkout counter or a self-serve car wash.

Chapter 13 illustrates the application of POAD in the development of a pattern-oriented design for the domain of digital-content processing and manipulation. This application is used to read, process, and handle digital content where heterogeneous source and delivery channels are supported, digital media is converted from one format to another, and metadata is extracted.

Chapter 14 illustrates the application of POAD in the development of a pattern-oriented design for part of a distributed medical informatics system that is based on the Digital Imaging and Communication in Medicine (DICOM) standard.

Part V discusses the automation of POAD and wraps up the discussion. Chapter 15 discusses the metamodeling support of UML semantics to POAD models. In Chapter 16 we discuss the tool support for applying, modeling, and composing design patterns. Chapter 17 discusses possible future trends that could build on top of the POAD methodology. Appendix A describes the pattern interfaces for some patterns that are used in the case studies in Part 4. Appendix B contains a discussion about the state of the art and practice in design patterns. The glossary provides the collection of terms used in the book, and finally a large selection of bibliographic information related to using patterns in software development is presented.

Sherif Yacoub and Hany Ammar
January 2003

Foreword

With the growing demand on rapid software development—to meet time-to-market needs—software development processes are shifting from the traditional development starting from scratch to reuse of existing solutions, whenever possible. With increases in the complexity of software systems, development from scratch has simply become an obsolete alternative. The question then becomes: what can we reuse and how can we reuse it? This book provides an answer to these questions: you can reuse design ideas and models in terms of design patterns, and you can reuse them in a systematic process called *Pattern-Oriented Analysis and Design* (POAD).

Design patterns and application frameworks are two essential technologies in developing complex software systems that are maintainable and stable (see *Building Application Frameworks: Object-Oriented Foundations of Framework Design*, Fayad et.al. 1999). Although black box component reuse and code reuse *are* effective techniques, practical experience shows that design reuse is equally if not *more* important. Design and architecture ideas stay with the system for its lifetime and hence an initial stable design increases chances of building a successful system.

No single pattern is used in isolation! A software system is a composition of multiple patterns and frameworks that are sometimes domain-specific and sometimes generic. Over the last decade, hundreds of software design patterns have been extracted from successful projects and documented. On the other hand, integrating those patterns together to develop application designs is far from complete. Application designers need design models that capture pattern compositions; they need systematic processes to guide them throughout the development process.

POAD is one successful methodology that provides a complete solution for composing design patterns. This book describes the detailed methodology. The key strengths of this book are:

- **Composition models:** it uses UML design models—mainly class and package diagrams—to illustrate how patterns can be glued together to create a sound design.
- **Processes**: it describes what the designer needs to accomplish at each phase—and hence provides a unique systematic approach to compose design patterns.
- **Case studies**: it illustrates POAD in action by describing four examples, including code samples.

The traceability among various designs models at different levels of abstraction—and linking all byproducts of the composition process—is another key differentiator in POAD. It is the practicality of the approach and its ease-of-use that makes it viable in real world application development.

Read carefully, try the examples, and apply them in your application designs. You will find it rewarding! Enjoy!

MOHAMED FAYAD
Professor, Computer Engineering, San Jose State University
CEO, ActiveFrameworks, Inc.

Contents