Undergraduate Topics in Computer Science

Undergraduate Topics in Computer Science' (UTiCS) delivers high-quality instructional content for undergraduates studying in all areas of computing and information science. From core foundational and theoretical material to final-year topics and applications, UTiCS books take a fresh, concise, and modern approach and are ideal for self-study or for a one- or two-semester course. The texts are all authored by established experts in their fields, reviewed by an international advisory board, and contain numerous examples and problems. Many include fully worked solutions.

For other volumes:
http://www.springer.com/series/7592

Sarnath Ramnath and Brahma Dathan

Object-Oriented Analysis and Design

Springer

Universities Press

Sarnath Ramnath
Department of Computer Science
St. Cloud State University
ECC 139
56303 St. Cloud, Minnesota
USA
rsarnath@stcloudstate.edu

Brahma Dathan
Department of Information
and Computer Science
Metropolitan State University
L118 New Main
700 7th Street East
55106 St. Paul, Minnesota
USA
Brahma.Dathan@metrostate.edu

A co-publication with the Universities Press (India) Pvt. Ltd, licensed for sale in all countries outside of India, Pakistan, Bhutan, Bangladesh, Sri Lanka, Nepal, The Maldives, Middle East, Malaysia, Indonesia and Singapore. Sold and distributed within these territories by the Universities Press (India) Private Ltd.

Undergraduate Topics in Computer Science ISSN 1863-7310
ISBN 978-1-84996-521-7 e-ISBN 978-1-84996-522-4
DOI 10.1007/978-1-84996-522-4
Springer London Dordrecht Heidelberg New York

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

Library of Congress Control Number: 2010934228

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Contents

Part II Introduction to Object-Oriented Analysis, Design, Implementation and Refactoring 111

Preface

At least some people reading the title of this book may wonder why there should be one more book on the topic of Object Oriented Analysis and Design (OOAD). The short answer to this question is that in our teaching of the subject for over a decade, we have not been able to find a suitable textbook on this topic at our respective universities.

We wrote up a long answer to the above question in a paper published in the 2008 SIGCSE conference. (So, if you are not satisfied with this preface, we hope you will consider reading our paper.) To summarise some of the observations and experiences in that paper, we note that our approach has always been to find ways to give a comprehensive introduction to the field of OOAD. Over the years the field has become quite vast, comprising diverse topics such as design process and principles, documentation tools (Unified Modelling Language), refactoring and, design and architectural patterns. In our experience, for most students the experience is incomplete without implementation, so, that is one more addition to the laundry list of topics to be covered in the course.

It was impossible to find a single book that gave a balanced coverage of all these topics in a manner that is understandable to the average college student. There are, of course, a number of books, some of them profound, that cover one or more of the above topics quite well. Besides their specialised nature, these books are primarily not meant to be textbooks. Expecting our students to read parts of these books and assimilate the material was not a realistic option for us.

This text is the result of our efforts over several years and provides the following:

1. A sound footing on object-oriented concepts such as classes, objects, interfaces, inheritance, polymorphism, dynamic linking, etc.
2. A good introduction to the stage of requirements analysis.
3. Use of UML to document user requirements and design.
4. An extensive treatment of the design process. The design step is, arguably, the most demanding activity (from an intellectual perspective) in the OOAD process. It is thus imperative that the student go through the design of complete systems. For pedagogical reasons we have kept the systems simple, yet sufficiently interesting to offer design choices. Going through these design exercises should help the student gain confidence to undertake reasonably complex designs.
5. Coverage of implementation issues. The reader will find critical excerpts from the implementation in Java. But he/she would be well advised to remember that this is not a book on Java. (More on this later.)
6. Appropriate use of design and architectural patterns.
7. Introduction to the art and craft of refactoring.

8. Pointers to resources that further the reader's knowledge.

It is important to remember what this book is *not* about.

1. It is not a book on Java. While the appendix has a short tutorial on the language and most of the code in the book is in Java, we do not cover constructs for the sake of teaching the language. Coverage is limited to the extent needed for understanding the implementation and for highlighting object-oriented concepts.
2. It does not cover software engineering concepts such as project management, agile technology, etc.
3. It does not treat UML extensively. Although we mention the various types of UML diagrams, many of them are not expanded because an occasion does not arise for such an undertaking.
4. It is not a catalog of design patterns or refactoring techniques. We cover only those patterns that arise naturally in our case studies. It has been our experience that design pattern discussions without a meaningful context are not well received by students.

Who will find this book useful?

Although the material in this text has primarily evolved out of a course taught for computer science senior undergraduates, others without a formal computer science background may also find this handy. In our program, students taking this are expected to have completed a course in data structures, but the material in this text does not require an intimate knowledge of the intricacies of any of these. A programmer who has used and is familiar with the APIs for some of the data structures could easily handle the material in the text. However, a certain amount of maturity with the programming process is needed, and for a typical undergraduate student this is usually obtained through a data structures course.

All the main case studies used for this book have been implemented by the authors using Java. The text is liberally peppered with snippets of code wherever we felt that a more 'concrete' feel for the design would be helpful. Most of these snippets are short and should be fairly self-explanatory and easy to read. Familiarity with a Java-like syntax and a broad understanding of the structure of Java would certainly be extremely helpful. The reader not familiar with Java but having significant software experience, need not, however, be deterred by this and can get a good feel of the entire OOAD process even without examining the code.

How to use this as computer science text

There clearly are several ways of structuring a computer science program, and the way in which this text could be used would depend on that structure.

The text is divided into three parts:

- **Part I** provides a thorough coverage of object-oriented ideas.
- **Part II** introduces the concepts of object-oriented analysis, design, implementation and, refactoring.
- **Part III** deals with more advanced design issues and approaches.

Part I, which comprises Chapters 1 through 4, gives a broad and solid foundation in concepts that are central to OOAD. The amount of time spent on covering these materials would vary considerably, depending on the program structure.

Part II begins in Chapter 5 with three useful design patterns. This part also includes Chapters 6 through 8, which introduces the first case study involving the analysis, design, and implementation of a simple library system. This is a critical choice since the entire process of design is being introduced through this case study. We chose this application because it met the following three major goals we had in selecting the case study: (i) the system should be simple so that it can be covered from analysis to implementation in a reasonable amount of time; (ii) students have an intuitive understanding of the application; (iii) several areas can be ‘naturally’ touched upon within the scope of the case study.

Several areas are touched upon in this case study and it would be pedagogically useful to emphasise these in the classroom.

- The importance of (and the quirks associated with) precisely specifying requirements and creating use case model.
- The design process. We naturally progress from the use case model to the the process of identifying classes and assigning responsibilities and coming up with sequence diagrams to implement use cases. The case study explores options in the design, which can result in lively discussions and contribute to student learning.
- The data is stored on stable storage so as to give students a sense of completeness. In this process, the student can see how the language quirks are affecting the implementation.
- The case study incorporates several design patterns in the code: Facade, Iterator, Adapter, Singleton, and Factory.
- Chapter 8 introduces refactoring and applies it to the completed design. This is done to underscore the fact that an awareness of refactoring is integral to the design process.

Covering this case study and assigning a similar project for students would be, in our opinion, essential. The amount of time spent on discussing these materials would depend on the background of the students.

Part III covers more advanced topics and spans Chapters 9 through 12. Chapter 9 introduces the use of inheritance in design, and also extends the case study. The use of inheritance was deliberately avoided in the main case study, not only to keep the case study simple, but also to ensure that the issues associated with the use of inheritance can be dealt with in context. The extension involves some inheritance hierarchies that allow us to illustrate sound object-oriented principles including the *Liskov Substitution Principle* and the *Open–Closed Principle.* A natural extension to the library system case study leads to a discussion of the Visitor pattern.

Chapter 10 deals with the second case study, which is from the domain of electronic devices that are controlled by software. Our example concerns a microwave oven that allows the user to perform the most common functions. To keep the case study manageable we have restricted the microwave functionality, but the model is enough for our purpose. Here we introduce the concept of states, finite state machines and state transition diagrams and compare and contrast it with the use case model. In this context, we introduce the State and Observer patterns.

The third case study, in Chapter 11, is an interactive program that can be used for creating figures. The objective here is to also examine the creation of larger systems that may require decomposition into subsystems. Before presenting the case study, the student is familiarised with the Model–View–Controller architecture. During the course of the case study, the student learns the Bridge, Command, and Composite patterns.

Chapter 12 shows how to design an object-oriented system for a distributed environment. As more and more applications become available remotely, we believe it is important for students to learn how to design and implement a distributed, object-oriented system. We have focused on Java Remote Method Invocation and the implementation of web-based systems using Java Servlets. To keep the discussion within reasonable size, we have left out other technologies such as ASP.NET and some important topics such as CORBA and distributed garbage collection.

Normally, while each case study is being discussed, we expect students to work on similar projects. This may be adapted as necessary to suit each situation. Presenting the topics in this integrated manner using case studies has been very helpful in giving students a complete picture of the OOAD process. We hope that by writing this textboot we have, in some small way, contribute to the advancement of the discipline.

Acknowledgments

The following individuals at Universities Press and Springer deserve special thanks: Madhu Reddy, Manoj Karthikeyan and Beverley Ford for help with the negotiations and the contract, and Sreelatha Menon for her efficient editorial work.

Brahma Dathan would like to thank his wife, Asha, and children, Anupama and Alok, for their support during the several years it took to complete this project.

Sarnath would like to thank his family, friends and colleagues for their encouragement and support during the years he worked on the project.

The authors would like to thank Dr Bina Ramamurhty for her helpful suggestions on an early draft of the book.

As we mentioned earlier, the book was shaped by our experience in teaching the subject over a fairly long period of time. Although the courses have stabilised now, the current form does not resemble much the original version taught a decade, or even four years ago. We experimented with the topics (adding, deleting, emphasising, de-emphasising and rearranging) and changed the pedagogical approach, moving from a theory-first-practice-later approach to a more case-study-based approach. Needless to say, we did all this at the expense of our students, but they took it all in good spirit. Many of our students also provided valuable, creative criticisms on different versions of the manuscript of the book. We cannot thank our students, past and present, enough!

Brahma Dathan
Sarnath Ramnath