# Book Reviews

Frederick P. Brooks, Jr.

## The Mythical Man-Month: Essays on Software Engineering

Addison–Wesley Publishing Company, Reading, Massachusetts, 1995, pp. xiii, 322, ISBN 0–201–83595–9

This is the anniversary edition of Brooks's authoritative book *The Mythical Man-Month* whose first edition appeared already in 1975. In it the author elaborates on the methodology of software development and tribulations of software project management based on his vast personal experience in this domain. In fact, the undertitle of the book — *Essays on Software Engineering* — is its best description, Brooks being concerned primarily with productivity in developing large software systems. The book is written very clearly and shows deep insight into this problem, using an informal style to present individual issues and related conclusions, thus making reading it a pure delight.

The book consists of three parts, each containing a group of essays. In its 15 chapters, the first part produces the text of the first edition. The central argument of this part is explicated especially in Chapters 2–7: the author notes that large programming projects suffer management problems different in kind from small ones, what is due to the division of labour. The critical need is preservation of conceptual integrity of the software system itself. These chapters explore both difficulties in achieving the indispensable unity, and the methods used. Other chapters explore different aspects of software engineering management.

The second part (Chapter 16) reprints the influential essay "No Silver Bullet: Essence and Accidents of Software Engineering", the 1986 IFIP paper that grew out of the author's experience chairing a Defence Science Board study on military software. It predicted that another decade would not see any programming technique that would by itself bring an order of magnitude improvement in software productivity. This work inspired a lot of discussion, and ten years later the readers themselves can conclude on the correctness of this assertion.

The third part consists of three chapters written especially for this edition, and mainly includes comments on published critique as well as author's reflections on past experience in software engineering. The last chapter is specifically an updating essay.

As the author noted, the book is intended for professional programmers, professional managers, and especially professional managers of programmers. It should be added that it is,beyond any doubt,a highly interesting reading for every computer professional.

*Uroš Peruško*
*Faculty of Electrical Engineering and Computing*
*University of Zagreb*
*Zagreb, Croatia*

Allen I. Holub

## Rules for C and C++ Programming

McGraw–Hill Book Company, New York, 1995, pp. 186, ISBN 0–07–029689–8

In spite of its folksy title, Holub's book is refreshingly oldfashioned, standing out from the current crop of thick verbose programming books, primarily in that:

- it is written by an intelligent and knowledgeable author, assuming intelligent readers with at least a basic command of the subject;
- it is not selling anything, except for the author's knowledge and wit;
- its points are put clearly and concisely, in a literate and often elegant language.

The result is 186 delightful pages, containing lots of information which is hard or impossible to find elsewhere. It is reminiscent of finest mathematical books, which are primarily about disciplined thinking, rather than about partial differential equations or algebraic fields: Holub's book is about disciplined thinking about C/C++ programs.

The book is organized into 'rules' which the author has found out to have worked well for him, ranging from very general points on software design and construction (and the organization of the process), to intricate details of C/C++ semantics and use. C and C++ are separate both spatially and conceptually — distinction between C and C++ approach is one of Holub's major points. It is not a textbook, and not a reference manual — its proper use seems to be to hang around, within reach from a programmer's desk or bedside, to be reopened from time to time.

Holub's opinions are strong, and sometimes highly personal, as he himself repeatedly stresses. The reader may thus often disagree, but, in this reviewer's opinion, she is well advised to take Holub seriously, and to explicate and analyze the points of disagreement, it makes very useful excercise. The reader would also be well advised to be a good sport and not to get offended by this or that insulting remark, since Holub hurls insults all around: at managers who measure productivity in lines of code, at doctors who never learn to use a pencil, at mathematicians who have written some of the worst programs ever, at programmers who can program Fortran in any language, at primadonnas who quickly produce unreadable programs which seem to work, at Microsoft whose MFC is a primary source of examples of how not to do things, at... In return, the reader gets a lot of good advice about and valuable insight into C/C++, and also splendid reading.

A short "did you know" quiz may help the reader determine his/her use for Holub's book. For C programmers: did you know that a cast in an expression in general involves a temporary variable of the target type? For C++ programmers: did you know that passing a class object by value, or returning it from a function, involves the copy constructor? If the (honest) answer is "yes of course", there is finally a book talking to you. If the answer is anything less than that, then you need Holub's book desperately.

*Dean Rosenzweig*
*Faculty of Mechanical Engineering*
*and Naval Architecture*
*University of Zagreb*
*Zagreb, Croatia*

Christopher W. Fraser, David R. Hanson

## A Retargetable C Compiler: Design and Implementation

The Benjamin/Cummings Publishing Company, Inc., Redwood City, USA, 1995, pp. xv, 564, ISBN 0–8053–1670–1

While most compiler texts focus on compiling algorithms and methods, which leaves room only for a toy compiler as an example, this book completely examines the design and implementation of `lcc`, a production-quality, retargetable compiler for the ANSI C programming language, designed at AT&T Bell Laboratories and Princeton University.

The aim of reading this book is to learn more about compiler construction. Not many programmers really need to know how to design and implement compilers, most of them work on applications and other aspects of systems programming. However, there are several reasons why even those C programmers may benefit from reading this book

First, programmers who understand how a particular compiler works are usually better programmers, since they gain knowledge about even the darkest corners of the programming language. Second, most texts on programming use only small, elegant examples to illustrate programming techniques. The majority

of programmers, however, work on very large programs, and thus encounter many problems evolving from the size of their programs. This book provides one such reference point, since it describes both the good and the bad points of `lcc`, a large and complex application. Third, a compiler is one of the best demonstrations of the interactions between theory and practice in computer science. Exploring these interactions in a real program helps programmers understand when, where and how to apply different techniques. Fourth, this book is an example of a "literate program", it presents `lcc`'s source code and the prose that describes it. The code is presented in the order that best suits understanding it, not in the order dictated by the C programming language.

A compiler generally translates source code to assembler or object code for a target machine. A retargetable compiler has multiple targets. `lcc` is a retargetable compiler, and this book tours not only most of the compiler itself, but also code generators for three target machines: MIPS R3000, SPARC, and Intel 386 and its successors (referred to as X86).

The book is organized as follows. The first chapter is an introduction, giving instructions to the reader on how to read the book. The second chapter describes `lcc`'s storage management scheme, which is based on object lifetimes. In this scheme allocation is very efficient and the cost of deallocation is negligible. It simplifies the code and makes allocation very cheap. The following chapter deals with symbol management in `lcc`. Symbols are collected into symbol tables, through which all parts of the compiler communicate. Chapter 4 focuses on types — representing types, type management, type predicates, type constructors, function types, structure and enumeration types, type-checking functions and type mapping.

The fifth chapter defines the interface between the target-independent front end and the target-dependent back ends of `lcc`. This interface consists of a few shared data structures, 19 functions and a 36-operator language which encodes the executable code from a source program in directed acyclic graphs, called dags. Chapter 5 has been made as self-contained as possible, allowing the readers to approach it independently from other chapters.

The lexical analyzer, which reads source text and produces token, the basic lexical units of the language, is described in Chapter 6. The lexical analyzer's main activity is moving characters, so minimizing the amount of character moving may help increase the speed. In `lcc`, this is done by dividing the lexical analyzer into two tightly coupled modules: the input module input.c and the recognition module `lex.c`. The described lexical analyzer provides a stream of tokens to the parser. `lcc` uses a recursive-descent parser, a straightforward application of classical parsing techniques for constructing parsers by hand, which produces a small and efficient compiler and is suitable for languages as simple as C or Pascal. The seventh chapter gives a theoretical background in formal language theory, syntax-directed translation and error handling, necessary for understanding the code described in subsequent chapters.

Chapter 8 focuses on C expressions. Those expressions form a sublanguage for which the parsing functions are relatively simple and straightforward to write. They are a good starting point for describing `lcc`'s eight modules that collaborate to parse and analyze the input program. This chapter concentrates on two of those modules: `tree.c` with low-level functions for managing trees and `expr.c` with parsing functions for recognizing and translating expressions.

Chapter 9 deals with expression semantics, describing semantic analyses that must be done to build trees for expressions. There are three separate problems with which these analyses must deal: implicit conversions, type checking and order of evaluation. Two modules involved in those analyses are `enode.c` with type-checking functions and `simp.c` with functions that perform tree transformations.

Statements are the subject of the tenth chapter. The semantics of statements consist of the evaluation of expressions, possibly intermixed with labels and jumps. Expressions are compiled into trees and then converted to dags. Jumps and labels are also described by dags. This chapter describes if statements, labels and gotos, loops, switch statements, return statements, and managing labels and jumps. The corresponding module is `stmt.c`.

The longest chapter of the book, Chapter 11, concentrates on declarations. Declarations

specify the types of identifiers, define structure and union types and give the code for functions. Declarations are the most difficult part of C to parse. This chapter describes how declarations are parsed and are internalized in the front end's data structures. It also covers function definitions, compound statements, finalizations and `lcc`'s main program. The modules involved are `decl.c`, `main.c` and `init.c`.

Chapter 12 describes the remaining missing pieces of `lcc`'s front end — those that convert trees to dags and append them to the code list, and the functions which back ends call from their interface procedures to traverse code lists. Those pieces appear in the module `dag.c`.

Chapters 13–18 cover the back ends of the `lcc` compiler. The thirteenth chapter is concerned with structuring of the code generator. The code generator supplies the front end with interface functions that find target-dependent instructions to implement machine-independent intermediate code. Interface functions also assign variables and temporaries to registers, to fixed cells in memory and to stacks, which are also in memory.

Selecting and emitting instructions is the subject of Chapter 14. The instruction selectors in this book are generated automatically from compact specifications by the program `lburg`, a code-generator generator. `lburg` accepts a compact specification and emits a tree parser written in C that selects instructions for a target machine.

The following chapter focuses on register allocation, which can be viewed as having two parts: allocation decides which values will occupy registers and assignment assigns a particular register to each value. The overall organization of the register allocator is described first, followed by tracking the register state, allocating registers and spilling.

Chapters 16, 17 and 18 present the modules that capture all information about three different targets — the MIPS R3000, SPARC and X86 (machines compatible with the Intel 386 architecture) architectures, respectively. Those chapters are concerned with registers, selecting instructions, implementing functions, defining data and copying blocks. A retrospective is given in the last, nineteenth chapter, followed by the bibliography.

This book is well suited for self-study by both professionals and academics. The readers are supposed to be fluent in C and some assembly language, to know what a compiler is and what a compiler does, and to have a working understanding of data structures and algorithms at the level covered in typical undergraduate courses. The book and the accompanying diskette offer complete documented source code for `lcc`, and the latest version of `lcc` is always available for anonymous `ftp` from `ftp.cs.princeton.edu` in `pub/lcc`.

*Andrea Budin*
*Faculty of Electrical Engineering and Computing*
*University of Zagreb*
*Zagreb, Croatia*

André Bacard

## The Computer Privacy Handbook

Peachpit Press, Berkeley, California, 1995, pp. xii, 274, ISBN 1–56609–171–3

Computers and computer communications have a great impact on our everyday life. A huge amount of information is stored on today's computers and shared between them. Some of it is about us and about the things we want to keep private. André Bacard, the author of this book, tries to show us how naive we are in our belief that our privacy cannot be easily invaded; however, he also suggests some steps we can take to protect our rights to privacy. To quote the author: "This book is a wake-up call... The goal of this book is to coax awareness. Life is just the right length for awareness. Those of us who are computer literate have the electronic power and the social responsibility to fight for our freedom of privacy".

The book is divided into six parts. Part I: *Guarding Our Privacy in the Information Age*, is an excellent introduction not only to the subject of privacy on the Internet but also to the whole subject of politics of privacy. Throughout the first part of the book there are lots of drastic examples from real life in which privacy of individual

persons has been neglected. The author's intention was to show us that there is already plenty of information about our private life that can be easily gathered and used to harm or control us. Medical and tax records, social security and telephone numbers as well as postal addresses are available not only to the government but also to everyone who can illegally access government, bank or hospital computers.

Part II: *Cryptology*, is an introduction to encryption techniques available today. This is by no means a mathematical introduction, but just an overview of the most widely used and publicly available techniques. This part deals with techniques such as: DES, RSA and IDEA and gives us a short historical background of the encryption technology. It also covers the well-known Clipper Chip Initiative, an attempt of the US government to popularize phone calls encryption with a special computer chip which would however allow US law enforcement agencies (and hopefully nobody else) to intercept voice communications.

Part III: *PGP: Pretty Good Privacy*, presents an easy-to-use, highly secure computer program called PGP, and written by Philip Zimmermann, that encrypts and decrypts data. A short history of PGP and a presentation of its author are covered along with his battle with the FBI (Federal Bureau of Investigation) and the NSA (National Security Agency) to allow PGP encryption for public use. Today, PGP is a freely available program ported to almost every computer architecture and operating system and is the de facto world standard software for E-mail security.

Part IV: *Using PGP on the PC*, is nothing but a classical user's guide for PGP written for PC DOS platforms. It covers both the installation and the usage of the program: how to create public and secret keys, and how to encrypt and decrypt messages and files, as well. There is a great deal of examples with accompanying computer screen images well suited for novice users.

Part V provides a bibliography to the subject of computer privacy. It can be used as a starting point for anyone who wants to know more about the subject. Part VI: *Pro-Privacy Cyberspace Resources*, lists resources, both on the Internet and elsewhere, that can help us to insure our privacy. There is also a list of non-government organizations, research projects, conferences, file archives and other Internet resources promoting privacy of individuals.

If you were not concerned about your privacy before, after reading this book you will start to ponder. This book also raises some interesting political questions. I myself recommend it to everyone using the Internet, so that he can find out how to protect his privacy while communicating with others. Part I of this book is an excellent piece of journalism written in plain language with a lot of humour and real word examples. I believe that you will read it in one breath.

*Goran Omrčen–Čeko*
*Faculty of Electrical Engineering and Computing*
*University of Zagreb*
*Zagreb, Croatia*

Tracy Laquey
Adapted by Richard Harris and Stephan Deutsch

## The European Internet Companion
A Beginner's Guide To Global Networking

Addison–Wesley Publishing Company, Wokingham, England 1995, pp. xiv, 272, ISBN 0–201–42778–8

This book has its origin in the best-selling introductory guide to the Internet, *The Internet Companion*, written in 1992 by Tracy Laquey. It is its adaptation for the European audience, hence the title *The European Internet Companion*. As expected from this kind of book, not only the hardcopy form exists, but also an online Internet Companion site at the Online BookStore (OBS). Throughout the book the democratic nature of the Internet is often emphasized. This political issue is best seen through the fact that the foreword is written by US Vice President Al Gore.

Although the book is intended for Internet neophytes, it offers plenty of useful information for users with certain Internet experience, too. In order to make the book more understandable the author gives comparisons with everyday life

and follows a didactic approach. The book contains a certain dose of fun and humor, necessary to keep the attention of a cover-to-cover reader. Practical issues are also suitably covered through numerous examples and sample commands to try.

The book consists of seven chapters and an appendix. The introductory Chapter 1 titled *What Is the Internet and Why Should You Know About It?* attempts to find a tentative answer to this question. In order to give a comprehensive response, the chapter overviews plenty of diverse aspects of the Internet. Briefly stated, it discusses Internet history, significance, success, nature, properties, benefits, influence, future and the like.

Chapter 2 explains some of the basic principles underlying the Internet. Starting with its history, it continues with a set of short guidelines to basic concepts and terms. It also reviews the Internet structure and organizations involved in its operation.

The third chapter deals with the many different means people use to communicate over the Internet. It covers e-mail and USENET News in depth, and briefly describes interactive types of communication (talk, Internet Relay Chat, CU-SeeMe and MBONE).

If you are interested in finding information on the Internet, Chapter 4 will instruct you in using online resources and services. It starts with basic, low-level class of services such as remote login and file transfer, and continues with information discovery and retrieval class of services like Archie, Gopher, WAIS and World Wide Web. Additionally, the second half of this chapter presents various client applications used in accessing the above services.

Throughout Chapter 5, the author covers several different topics which are denoted as advanced. The beginning section contains accounts on Internet culture, myths, legends and even games, and is followed by a section that provides an overview of security issues additionally furnishing some related recommendations. Next comes a section on organizations commited to the Internet. The chapter ends with a description of services and methods for finding e-mail addresses and resources.

To provide a more complete reference in using the Internet, the author devoted Chapter 6 to an introduction to UNIX, since many computers on the Internet run it. Except for the basic commands, it also provides explanations of the most common UNIX idiosyncrasies and applications.

The final chapter discusses how to connect to the Internet, what is needed to start the communication and where to go for the required services. It also comments the choices for individual access as well as basics for connecting business organizations.

The Appendix is intended to be a sort of summary. The information presented there mostly apply to the European market, since the book is addressed to this audience. It lists service providers by countries (I noted the absence of Croatia), selected resources for navigating, organizations and consortiums, commercial information services, client software and tools.

*Joško Poljak*
*Faculty of Electrical Engineering and Computing*
*University of Zagreb*
*Zagreb, Croatia*

Donald K. Burleson

## Managing Distributed Databases. Building Bridges between Database Islands

A Wiley/QED Publication, John Wiley & Sons, Inc., New York, 1994, pp. 367, ISBN 0–471–086223–1

In this book Donald K. Burleson provides solutions to every conceivable problem associated with the management of distributed database systems, including those in client/server environments.

"Open systems" has become one of the foremost buzzwords of the 1990s, but very little has been written about issues involved in managing a diverse network of databases and making disparate database systems behave as parts of an integrated enterprise. The problems of managing distributed database systems

that span geographical areas, hardware platforms and database architectures are very complicated. Distributed database expert Donald K. Burleson covers in 17 chapters many important problems and provides a detailed overview of the technological and human issues involved in creating and maintaining a successful distributed database.

The basic terms and historical overview about databases are addressed at the beginning of this book: history of distributed databases (Chapter 1), what is a database (Chapter 2), overview of distributed databases (Chapter 3), and basics of client/server systems (Chapter 4). The economics and human factor in distributed database design are presented in Chapter 5. Chapter 6 deals with enterprise management of distributed databases. Chapters 7 and 8 explain the influence of object technology to distributed databases. COBRA — the common architecture for distributed systems is presented in Chapter 9. The next five chapters cover other distributed database related problems: distributed database connectivity (Chapter 10), database access control and connectivity (Chapter 11), distributed database backup and recovery (Chapter 12), distributed database security (Chapter 13), and performance and tuning for distributed databases (Chapter 14). New data retrieval methods are explained in Chapter 15. The remaining two chapters deal with human issues: psychological issues — dealing with resistance (Chapter 16) and new human roles in distributed database systems (Chapter 17). Main products for creating object oriented distributed systems that are the result of numerous projects are described in the appendix. The book has a very rich bibliography and a well-structured index.

This book *Managing Distributed Databases. Building Bridges between Database Islands* gives an excellent explanation of all important problems related to distributed databases. The book possesses no deep theoretical background, however it includes a comprehensive and fascinating set of examples in a variety of language, management and application domains. Its main feature is a clear definition and explanation of a large number of different terms related to the domain of distributed databases. It can be seen that the author is an acknowledged expert who has worked in the consolidation of databases on a variety of platforms and architectures. Moreover, the writing style of the book is clear and acceptable even for readers with a limited background in distributed database. On the whole, this is a book which I enjoyed reading very much and I recommend it to anyone interested in the practice of distributed databases.

*Zoran Skočir*
*Faculty of Electrical Engineering and Computing*
*University of Zagreb*
*Zagreb, Croatia*

## D. Collins

## Designing Object-Oriented User Interfaces

Benjamin/Cummings Publishing Company, Inc., Redwood City, CA, 1995, pp. xvi, 590, ISBN 0–8053–5350–X

The book *Designing Object-Oriented User Interfaces* by Dave Collins investigates user interface design, a computer science area which is nowadays increasingly showing its importance. It studies this area through object orientation, a procedure which, besides providing a closer and very convenient link with a (still) modern and promising software engineering methodology — the one of object oriented development, underlines deeper conceptual ties to be exploited in user interface design.

The book is structured into three parts: a *Foundation* within its roughly 80 pages provides introductory matter on user interfaces (UIs) and on applying object orientation onto it, *External Design* (about 200 pages) delivers an account on UI design methodology, encompassing important items as the conceptual model and the "look and feel", i.e. the presentation and action languages, while *Internal Design* (slightly less than 200 pages) details the general software architectures, methods and tools used in object oriented UIs (OOUIs) and corroborates this with examples.

The exposition along the chapters proceeds as follows. Chapter 1 introduces definitions of UI

and its treatment in the book, the book's general plan, the audience it addresses, and the relationship to other methods/books on UI. The three chapters of Part I present the framework for understanding OOUI. In *Evolution of the OOUI*, an analysis of characteristics of end users and their work along with a historical overview of UIs, stressing Xerox PARC seminal influence on the field, is given. Chapter 3 describes the two most frequently encountered UI styles: the character-based and the graphical one. A "naive" analysis is provided, still not directly relying on OO principles, yet exposing the basic issues of user perception. Next, the application of object orientation to UI external characteristics is introduced, identifying elements of the OO paradigm (OOP) along with the definition of OOUI itself.

Part II in its 6 chapters documents methodology and techniques for concrete and observable UI objects design. In Chapter 5 the three domains of OO design for UI are presented: system conceptual model, representation of objects provided by the UI, and information structures implementing the objects' functional characteristics. Their description, the relevance to UI development, and respective illustration through the well known Solitaire game is provided, too. The following chapter brings an exposition of basic software engineering development models along with a model for the OOUI design process. This model is based on an interrelated iteration of three activity clusters — conceptual model (metaphor) definition, conceptual model objects actualization, and testing and evaluation. Chapter 7 underlines the importance of understanding users, their tasks and work environments, and delves deeper into task analysis, including its documenting, levels of detail and interpretation of tasks within OOP. In *The User's Conceptual Model* the terminology of models and metaphors with accompanying examples is given, as well as a method for user conceptual model design and evaluation, supported by a catalog of UI metaphors for design ideas generation. The next chapter is devoted to presentation of information onto computer displays (view) with particular attention on the use of OO principles. A step-by-step method for designing OOUI appearance is offered, and corroborated with content view and icon design. The chapter *Interaction and Control Mechanisms* provides a general framework for reasoning about interaction. Interactivity is considered as a cyclic activity resulting in information presentation as a response to user action, and therefore determining the "look and feel" of a UI design. Nature of interactivity, review of interaction devices, step-by-step approach to designing OOUI "feel", style guides in OOUI design, and documenting UI "look and feel" design are some of the most relevant topics addressed.

Finally, in the third part (Chapters 11–16) aspects of designing the implementation model that are important to UI designers are discussed. *Object-Oriented System Architectures* brings the OO view of systems and applications. The two influential UI implementation models — language, encompassing command and menu-based UIs, and event, combining primitive events to create new ones, are described, the latter as the basis for commercial windows systems. After overviewing (event-driven) windows managers, the following topics make the rest of the chapter: UI code structuring, MVC (mental-view-controller) model as a software engineering technique, key tools for modularity in UI implementation, and model-view separation as the basis for well-structured implementations. Chapter 12 provides information models as concrete views of users' conceptual models. Aspects of information models such as protocols supporting visible UIs, general structuring issues in designing information models, "infrastructures" for building information models from OO and non-OO components are also worked out in the chapter. Chapter 13 highlights software structure aspects that promote interactivity: presentation objects — views implementing the presentation language, and interaction objects — controllers or views behaving as virtual devices and affording some user action that implements the action language. After a more general discussion on interaction and feedback, direct manipulation (DM), an interaction technique using displayed objects as "input devices" to control the UI, is elaborated, here including DM protocols encapsulation methods for making them simpler and more reusable. In Chapter 14 OO tools for prototyping and implementation are recapitulated, along with related issues like tool requirements, tool components and portability, their classification and evaluation. In the subsequent Chapter 15 some OOUI design examples are illustrated among which a seamless

online news system (Journalist), a distributed multimedia system (KCM, Knowledge and Colaboration Machine), and a virtual reality toolkit (VR-DECK, Virtual Reality Distributed Environment and Construction Kit). The last chapter gives a summary of the most important points in the book, reviews OOUI current state of the art and discusses presently detectable trends for future OOUIs.

The book is complemented by two appendices treating a fax case study, and an introduction to object orientation. A nice Glossary with some 100 entries offers a short explanation of relevant acronyms and terms, while the impressive Bibliography, intended primarily as a guide to resources for further study, lists nearly 500 entries for books, papers, and articles, 7 entries for videotapes and somewhat less than 40 entries for software packages. The well compiled Index extends over 18 pages.

As in the case of other Benjamin/Cummings editions, this book is superbly produced. The structure is impeccable and is based upon the development process so enabling quick focusing on desired subject matter. The book's three parts are well-chosen and the chapter organization is performed accordingly well. It brings a lot of subject matter on a computer science area whose importance is increasingly growing, and exposes it by linking it to a modern and promising software engineering methodology. Moreover, there is a wealth of examples, exercises and very helpful suggestions for individual study. As stated in the book, it is intended for working developers and students of interface design. In my opinion it has accomplished this task outstandingly.

*Vlado Glavinić*
*Faculty of Electrical Engineering and Computing*
*University of Zagreb*
*Zagreb, Croatia*