Book Reviews

Martin D. Carroll, Margaret A. Ellis

Designing and Coding Reusable C++

Addison–Wesley Publishing Company, Reading, Massachusetts, 1995, pp. 317, xvi, ISBN 0–201–51284–X

Most professional programmers are well acquainted with the problem of code reusability. Libraries with standard functions are available for all programming languages, but writing libraries of one's own, in such a way that they could be used later in many different projects, requires the programmer's thorough approach. This particularly holds for C++ programming language.

Over the last few years, C++ has become the most popular programming language. It gained popularity mainly from the widespread C programming language, from which it has evolved. One of the qualities of C is availability of a huge function library. But, as the reader probably knows, C++ is not just a "better C" - it extends C in several ways, providing object-oriented design and programming. Some of the features added to C++ provide better code reusability, templates and inheritance being the most obvious ones. But, these built-in features cannot solve all reusability-related problems. The problem of reusability is emphasized by the fact that C++ is not standardized yet and there is an obvious problem of code porting between compilers. The authors of the book, M. D. Carroll and M. A. Ellis have been directly involved in the design and development of C++ language and C++ compilers at AT&T Bell Laboratories. Moreover, Ms. Ellis is the co-author of *The Annotated C++ Reference Manual*, first C++ *de facto* standard. Mr. Carroll has worked on the design and implementation of reusable C++ libraries for several years.

Designing and Coding Reusable C++ is divided in twelve chapters through which the authors analyze almost all aspects of code reusability. Each chapter closes with a summary, exercises and references for further reading. The book ends with an extensive bibliography and alphabetical index.

In the introductory chapter, basic facts regarding the code reusability are discussed. Some myths of reuse are repelled and obstacles are outlined. There follows a chapter on class design. Since most of C++ libraries are primarily collections of classes, a good class design can establish satisfactory reusability. Several topics essential for design of reusable classes are covered, e.g. abstraction, regular functions, class interface consistency, conversions. A special attention is paid to shallow and deep copies, and to the use of *const*.

The *Extensibility* chapter covers mainly inheritance, one of the most powerful features of C++. It is followed by a chapter on efficiency, an essential property of the reusable code. The term efficiency here does not mean the run time efficiency only, but also covers the program build time (including compile, link and instantiation times), code size and memory. Special attention is paid to function inlining.

Since the occurrence of errors during the execution of programs is unavoidable, in order to achieve a complete reusability, library code must deal with errors reasonably. Accordingly, the problem of error detection and handling is described in the fifth chapter, with particular emphasis on exception handling - a mechanism introduced to C++ to efficiently manage anomalies during program execution.

The *Conflict* chapter is dealing with a very common problem of name conflicts. Very often, same names are used to mean different things in different libraries. If these names have global scope, they will conflict with each other. Such name conflicts occur between macro names and environment names as well. The authors propose a naming convention that should avoid such conflicts. Moreover, a namespace construct is described, which has been included in ANSI/ISO C++ standard lately in order to circumvent name conflicts in large-scale projects.

The seventh chapter examines the compatibility problem, namely the source, link and run compatibility. The importance of backward compatibility in future releases is pointed out. The *Inheritance Hierarchies* chapter studies rootedness, depth, and fanout of inheritance hierarchy. In this chapter the reader will find a very useful discussion on the permanent dilemma “Templates or inheritance?”.

Portability, closely related to code reusability, is discussed in chapter 9. Major issues that affect portability are reviewed, e.g. the changing definition of the C++ language, undefined behaviors, template instantiation, run-time libraries. There follows a chapter in which authors discuss whether a C++ library should reuse part of another library. The main drawbacks of using other libraries are designated and self-contained libraries, as an alternative to reusing other libraries, are described.

As pointed out in chapter 10, writing good documentation is particularly difficult but may be crucial for the success of any library - the authors give suggestions for writing tutorials and reference manuals. In the closing chapter several miscellaneous topics important for library reusability are examined. These topics include static initialization problem, endogenous and exogenous classes, iterators.

As the authors state, “. . . the primary goal of this book is to show how to write reusable code. . . - that is, code that can easily be used with little or no change in five, fifty or five hundred programs with varying requirements, written by different programmers, and possibly running on different systems. It is not a goal of this book to argue that all codes should be reusable. . . ”. A very instructive and systematic approach to topics throughout the chapters makes the matter appropriate for any programmer familiar with language basics. Topics are regularly illustrated with source code examples.

Writing reusable C++ code is not a trivial task, but the efforts will pay well, giving rise to productivity. Importance of the matter covered and the above mentioned qualities of the book make it therefore unavoidable for anyone who is or intends to become a professional C++ programmer.

Julijan Šribar
Faculty of Electrical Engineering and Computing
University of Zagreb
Zagreb, Croatia

Murray Sargent III, Richard L. Shoemaker

The Personal Computer from the Inside Out

Third Edition, Addison–Wesley Publishing Company, Reading, Mass., 1995, pp. xvi, 800, ISBN: 0–201–62646–2

The Personal Computer from the Inside Out provides a comprehensive treatment of the low-level PC hardware and software. The book consists of thirteen chapters.

The introducing Chapter 1 describes the evolution of the PC from the initial IBM PC and PC-AT machines to much more powerful PCs. It also gives an overview of the PC hardware and software. Chapter 2 gives an introduction to the basic concepts and terminology of machine and assembly language in terms of the x86 microprocessor's instruction set, and shows how a modern computer works internally as it performs basic operations. Chapter 3 discusses assembly language and the x86 instruction set in greater detail. The usage of various instructions and a macro assembler in writing simple programs are also described. The following advanced assembly language techniques are described in Chapter 4: how to structure complex assembly language programs, how to use macros and conditional assemblies, how to write assembly language subroutines for programs written in various high-level languages, how to write custom driver programs for any device attached to the computer, how to control disk files and how to program the floating-point unit. Chapter 5 describes the protected, virtual-address mode of the 286, 386, and later microprocessors. It also compares this to the 8086 real-address mode, virtual 8086 mode and the "real-big mode".

Chapter 6 provides an introduction to the basic digital circuitry, including diodes and transistors, simple logic gates, buffers, flip-flops, latches, clock circuits, counters, shift registers, multiplexers and demultiplexers and programmable logic devices, all of which are essential in building a microcomputer like the PC or in designing custom PC interfaces. These topics are expanded in Chapter 7, which deals with the PC hardware, such as the x86 microprocessor itself, memory, interrupt controller, RTC circuit, DMA controller, keyboard controller and expansion buses (ISA, VL, PCI). The concepts of interrupts, timer software and data transfer techniques are discussed in Chapter 8.

Chapter 9 shows how the PC's keyboard and video displays work and in some detail discusses the most popular video display options for the PC. Chapter 10 discusses how the TTL level highs and lows of I/O ports can be connected to various devices that manipulate and measure a variety of real-world quantities such as switch and relay closures, analog signal inputs and outputs, electrical waveforms, and motor control signals. The solutions to various problems encountered in acquiring and generating waveforms are also discussed, including the role of DSPs in waveform processing and manipulation. Chapter 11 explains various parallel-port and serial-port protocols and their usage on the PC. This chapter also discusses the IEEE-488 Interface Bus and the SCSI bus. Chapter 12 describes various media and methods for the storage of computer data in PCs. The approaches vary from small high-speed cache RAM right on the CPU chip itself, to inexpensive, large capacity, removable optical storage. The closing Chapter 13 gives useful tips for designing, constructing and debugging user interfaces, using a wire-wrapped analog and digital I/O board as an example project.

The book is written clearly and comprehensively, the matter is well- structured and it is supplemented with an index. Although this book offers a great deal of technical information, the authors succeeded in keeping it readable. I mostly appreciated the accompanied disk with the SST debugger, which is used to illustrate many principles of assembly- language programming and much of the inner workings of the PC. This book is useful for anyone who wants a thorough understanding of how PC and other computers work, and for anyone who wants to develop hardware or software extensions for the PC. If you consider yourself to belong to this group, you should certainly have a look.

Ninoslav Matić
Faculty of Electrical Engineering and Computing
University of Zagreb
Zagreb, Croatia

C. Mazza, J. Fairclough, B. Melton, D. De Pablo, A. Scheffer, R. Stevens, M. Jones, G. Alvisi

Software Engineering Guides

Prentice Hall International (UK), Hertfordshire, 1996, pp. x, 544, ISBN 0–13–449281–1

This book contains guides to software engineering produced by the European Space Agency (ESA) and is intended mainly for software developers applying ESA's Software Engineering Standards (PSS-05-0), which have been published in the book *Software Engineering Standards* (Prentice Hall, 1991). The *Software Engineering Standards* define the mandatory and recommended practices for specifying, developing and maintaining software and the *Software Engineering Guides* in this book discuss those practices in more detail.

According to the *Software Engineering Standards*, the software engineering activity may be divided into two parts: the products themselves and the procedures used to make them. The process of production is partitioned into six phases. This book gives a thorough guide for each of them.

The first phase of the software life cycle is the User Requirements Definition Phase (UR phase), which may also be referred to as the problem definition phase. This phase refines an idea about the task that has to be performed by computing equipment into a definition about what is expected from the computer system. The second phase is the Software Requirements Definition Phase (SR phase), which can be called the problem analysis phase. Based on the analysis of the user requirements, complete, consistent and correct software requirements are produced. What follows is the Architectural Design Phase (AD phase), the third phase of the software life cycle, which can also be named the solution phase. It defines the software in terms of the main software components and interfaces. The following, fourth phase is the Detailed Design and Production Phase (DD phase) - the implementation phase. This is the phase in which developers code, document and test the software after detailing the design specified in the AD phase. The fifth phase is the phase in which the developers release the software to the users. This is the handover phase, in Software Engineering Standards defined as the Transfer Phase (TR phase). The software is installed on target computer systems and acceptance tests are run to validate it. The last phase of the software life cycle is the Operation and Maintenance Phase (OM phase). This is the operational phase in which software is operated by the users and the end products and services it provides are utilized.

The guides for each phase give a detailed overview of the phase, present the methods as well as adequate tools, which can be used in the phase. Moreover, guidelines for the appropriate documentation of each phase are given. Each phase has its own index and a glossary, consisting of a list of acronyms and, in some phases, of a list of terms. References containing relevant material are given in each guide. Each guide is equipped with mandatory practices, and some, as a helpful hint contain an example of a requirements traceability matrix, CASE tool selection criteria, etc.

The second part of the book contains guides for procedures to be followed in the software life cycle, which are divided amongst four management activities. The first guide describes activities of planning, organizing, staffing, leading, monitoring and controlling a software project. Those activities are called Software Project Management (SPM).Each project must have a Software Project Management Plan in which its activities are defined in detail. The SPM guide should be read by software project managers, software quality assurance managers, senior managers, initiators of software projects and team leaders. The Software Configuration Management (SCM) is the activity of controlling the documentation and code throughout the software life cycle. Those activities must be defined in a Software Configuration Management Plan. The SCM guide is intended for those concerned with developing, installing and changing software - software project managers, software librarians and software engineers. The third group of activities are the activities of software verification during each phase of its development life cycle and of software validation when it is transferred. Those activities are called the Software Verification and Validation activities (SVV) and they must be defined in a Software Verification and Validation Plan. The guide for those activities should be read by everyone concerned with developing software. Finally, the *Software Engineering Standards* require that all software projects assure that product and procedures conform to standards and plans. This group of activities is called Software Quality Assurance (SQA) and must be defined in a Software Quality Assurance Plan. The SQA guide is intended for anyone dealing with software quality.

Each of the above guides contains an overview of activities, guidelines on methods and tools to be used, as well as instructions for planning. Each guide is equipped with a glossary, a list of references and mandatory practices. What many of the readers may find very useful are form templates for the documentation of Software Project Management and Software Configuration Management activities. The primary source of terminology and definitions of products and plans throughout this book are the software engineering standards of the Institute of Electrical and Electronic Engineers (IEEE).

This is book is a necessary companion for all those who want to apply effectively the *Software Engineering Standards* of the European Space Agency. However, it is also strongly recommended to anyone else interested in gaining an insight of the tedious tasks and steps in the development of a large software project, in order to make those tasks and steps easier, more formal and more effective.

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