

# Object Oriented Programming with C++ Fourth Edition

## C++面向对象程序设计 (第4版)

E Balagurusamy 著



大学计算机教育国外著名教材系列（影印版）

# Object Oriented Programming with C++

Fourth Edition

## C++面向对象程序设计

（第4版）

E Balagurusamy



清华大学出版社  
北 京

E Balagurusamy

**Object Oriented Programming with C++, Fourth Edition**

EISBN: 0-07-066907-4

Copyright © 2009 by The McGraw-Hill Companies, Inc.

Original language published by The McGraw-Hill Companies, Inc. All Rights reserved. No part of this publication may be reproduced or distributed by any means, or stored in a database or retrieval system, without the prior written permission of the publisher.

Authorized English language edition jointly published by McGraw-Hill Education (Asia) Co. and Tsinghua University Press. This edition is authorized for sale only to the educational and training institutions, and within the territory of the People's Republic of China (excluding Hong Kong, Macao SAR and Taiwan). Unauthorized export of this edition is a violation of the Copyright Act. Violation of this Law is subject to Civil and Criminal Penalties.

本书英文影印版由清华大学出版社和美国麦格劳-希尔教育出版(亚洲)公司合作出版。此版本仅限在中华人民共和国境内(不包括中国香港、澳门特别行政区及中国台湾地区)针对教育及培训机构之销售。未经许可之出口,视为违反著作权法,将受法律之制裁。

未经出版者预先书面许可,不得以任何方式复制或抄袭本书的任何部分。

北京市版权局著作权合同登记号 图字: 01-2009-4350 号

本书封面贴有 McGraw-Hill 公司防伪标签,无标签者不得销售。

版权所有,侵权必究。侵权举报电话: 010-62782989 13701121933

**图书在版编目(CIP)数据**

C++面向对象程序设计: 第4版 = Object Oriented Programming with C++, Fourth Edition: 英文 / (印) 巴拉古路萨米 (Balagurusamy, E.) 著. —北京: 清华大学出版社, 2009.9

(大学计算机教育国外著名教材系列)

ISBN 978-7-302-20732-0

I. C… II. 巴… III. C 语言—程序设计—高等学校—教材—英文 IV. TP312

中国版本图书馆 CIP 数据核字 (2009) 第 144171 号

责任印制: 孟凡玉

出版发行: 清华大学出版社

地 址: 北京清华大学学研大厦 A 座

<http://www.tup.com.cn>

邮 编: 100084

社 总 机: 010-62770175

邮 购: 010-62786544

投稿与读者服务: 010-62776969, [c-service@tup.tsinghua.edu.cn](mailto:c-service@tup.tsinghua.edu.cn)

质 量 反 馈: 010-62772015, [zhiliang@tup.tsinghua.edu.cn](mailto:zhiliang@tup.tsinghua.edu.cn)

印 装 者: 北京鑫海金澳胶印有限公司

发 行 者: 全国新华书店

开 本: 185×230 印张: 40.75

版 次: 2009 年 9 月第 1 版

印 次: 2009 年 9 月第 1 次印刷

印 数: 1~3000

定 价: 66.00 元

本书如存在文字不清、漏印、缺页、倒页、脱页等印装质量问题,请与清华大学出版社出版部联系调换。联系电话: 010-62770177 转 3103 产品编号: 034369-01

# 出版说明

进入 21 世纪,世界各国的经济、科技以及综合国力的竞争将更加激烈。竞争的中心无疑是对人才的竞争。谁拥有大量高素质的人才,谁就能在竞争中取得优势。高等教育,作为培养高素质人才的事业,必然受到高度重视。目前我国高等教育的教材更新较慢,为了加快教材的更新频率,教育部正在大力促进我国高校采用国外原版教材。

清华大学出版社从 1996 年开始,与国外著名出版公司合作,影印出版了“大学计算机教育丛书(影印版)”等一系列引进图书,受到国内读者的欢迎和支持。跨入 21 世纪,我们本着为我国高等教育教材建设服务的初衷,在已有的基础上,进一步扩大选题内容,改变图书开本尺寸,一如既往地请有关专家挑选适用于我国高校本科及研究生计算机教育的国外经典教材或著名教材,组成本套“大学计算机教育国外著名教材系列(影印版)”,以飨读者。深切期盼读者及时将使用本系列教材的效果和意见反馈给我们。更希望国内专家、教授积极向我们推荐国外计算机教育的优秀教材,以利我们把“大学计算机教育国外著名教材系列(影印版)”做得更好,更适合高校师生的需要。

清华大学出版社

# Preface

Object-Oriented Programming (OOP) has become the preferred programming approach by the software industries, as it offers a powerful way to cope with the complexity of real-world problems. Among the OOP languages available today, C++ is by far the most widely used language.

Since its creation by Bjarne Stroustrup in early 1980s, C++ has undergone many changes and improvements. The language was standardized in 1998 by the American National Standards Institute (ANSI) and the International Standards Organization (ISO) by incorporating not only the new features but also the changes suggested by the user groups. This book has been thoroughly revised and this edition confirms to the specifications of ANSI/ISO standards. Besides confirming to the standards, many smaller changes and additions to strengthen the existing topics as well as corrections to typographical errors and certain inaccuracies in the text have been incorporated. The highlight of this edition is the inclusion of two new programming projects in Appendix A - (1) Menu Based Calculation System and (2) Banking System that demonstrate how to integrate the various features of C++ in real life applications.

This book is for the programmers who wish to know all about C++ language and object-oriented programming. It explains in a simple and easy-to-understand style the what, why and how of object-oriented programming with C++. The book assumes that the reader is already familiar with C language, although he or she need not be an expert programmer.

The book provides numerous examples, illustrations and complete programs. The sample programs are meant to be both simple and educational. Wherever necessary, pictorial descriptions of concepts are included to improve clarity and facilitate better understanding. The book also presents the concept of object-oriented approach and discusses briefly the important elements of object-oriented analysis and design of systems.

## Key Pedagogical Features

### Key Concepts

Key concepts provide a quick look into the concepts that will be discussed in the chapter. These are followed by an Introduction that introduces the topics to be covered in that chapter and also relates them to those already learned.

### Key Concepts

- Software evolution
- Procedure-oriented programming
- Object-oriented programming
- Objects
- Classes
- Data abstraction
- Encapsulation
- Inheritance
- Polymorphism
- Dynamic binding
- Message passing
- Object-oriented languages
- Object-based languages

### 1.1 Software Crisis

Developments in software technology continue to be dynamic. New tools and techniques are announced in quick succession. This has forced the software engineers and industry to continuously look for new approaches to software design and development, and they are becoming more and more critical in view of the increasing complexity of software systems as well as the highly competitive nature of the industry. These rapid advances appear to have created a situation of crisis within the industry. The following issues need to be addressed to face this crisis:

- How to represent real-life entities of problems in system design?
- How to design systems with open interfaces?

completed before the next begins. The activities include problem definition, requirement analysis, design, coding, testing, and maintenance. Further refinements to this model include iteration back to the previous stages in order to incorporate any changes or missing links. *Problem Definition*: This activity requires a precise definition of the problem in user terms. A clear statement of the problem is crucial to the success of the software. It helps not only the developer but also the user to understand the problem better.

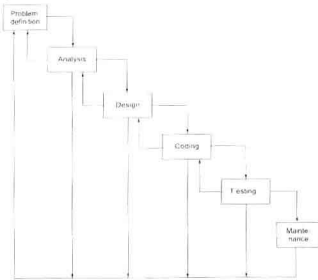


Fig. 17.2 6.3 Classic software development life cycle (Embedded 'water-fall' model)

## Figures

Figures are used exhaustively in the text to illustrate the concepts and methods described.

## Program Codes

Codes with comments are given throughout the book to elaborate how the various lines of code work.

### OBJECTS AS ARGUMENTS

```
#include <iostream>

using namespace std;

class time
{
    int hours;
    int minutes;
public:
    void gettime(int h, int m)
    { hours = h; minutes = m; }
    void puttime(void)
    {
        cout << hours << " hours and ";
        cout << minutes << " minutes " << "\n";
    }
    void sum(time, time); // declaration with objects as arguments
};

void time::sum(time t1, time t2) // t1, t2 are objects
{
    minutes = t1.minutes + t2.minutes;
    hours = minutes/60;
    minutes = minutes%60;
    hours = hours + t1.hours + t2.hours;
}

int main()
{
    time T1, T2, T3;

    T1.gettime(2,45); // get T1
    T2.gettime(3,30); // get T2

    T3.sum(T1,T2); // T3=T1+T2

    cout << "T1 = "; T1.puttime(); // display T1
    cout << "T2 = "; T2.puttime(); // display T2
    cout << "T3 = "; T3.puttime(); // display T3

    return 0;
}
```

PROGRAM 5.7

The output of Program 2.3 is:

```
Enter Name: Ravinder
Enter Age: 30

Name: Ravinder
Age: 30
```

### note

`cin` can read only one word and therefore we cannot use names with blank spaces.

The program defines **person** as a new data of type class. The class **person** includes two basic data type items and two functions to operate on that data. These functions are called **member functions**. The main program uses **person** to declare variables of its type. As pointed out earlier, class variables are known as **objects**. Here, **p** is an object of type **person**. Class objects are used to invoke the functions defined in that class. More about classes and objects is discussed in Chapter 5.

## Notes

Language tips and other special considerations are highlighted as notes wherever essential.

## Summary

Summary gives the essence of each chapter in the form of bulleted points which will be helpful for a quick review during the examinations.

### SUMMARY

- ❖ C++ is a superset of C language.
- ❖ C++ adds a number of object-oriented features such as objects, inheritance, function overloading and operator overloading to C. These features enable building of programs with clarity, extensibility and ease of maintenance.
- ❖ C++ can be used to build a variety of systems such as editors, compilers, databases, communication systems, and many more complex real-life application systems.
- ❖ C++ supports interactive input and output features and introduces a new comment symbol `//` that can be used for single line comments. It also supports C-style comments.
- ❖ Like C programs, execution of all C++ programs begins at `main()` function and ends at `return()` statement. The header file `iostream` should be included at the beginning of all programs that use input/output operations.

## Key Terms

Key terms listed in each chapter give the list of important terms discussed in the chapter.

### Key Terms

- Abstract base classes
- 'address of' operator
- argument object
- arrays of pointers
- arrow operator
- base address
- base object
- base pointer
- call back function
- class hierarchy
- compile time
- compile time polymorphism
- dereference operator
- Derived object
- do-nothing function
- dot operator
- dynamic binding
- early binding
- function overloading
- function pointer
- Implicit argument
- indirection operator
- invoking object
- late binding
- **new** operator
- Null pointers
- object pointer
- operator overloading
- placeholder
- pointers
- pointer arithmetic
- pointers to functions
- polymorphism
- pure virtual function
- run time
- run time polymorphism
- static binding
- static linking
- **this** pointer
- virtual constructors
- virtual destructors
- virtual function
- void pointers

### Review Questions

- 9.1 What does polymorphism mean in C++ language?
- 9.2 How is polymorphism achieved at (a) compile time, and (b) run time?
- 9.3 Discuss the different ways by which we can access public member functions of an object.
- 9.4 Explain, with an example, how you would create space for an array of objects using pointers.
- 9.5 What does **this** pointer point to?

## Chapter-end Exercises

More than 350 chapter-end exercise problems are given for the students to work out and practice. These include review questions, debugging exercises and programming problems.

## Programming Projects

The two programming projects in Appendix A will give an insight on how to integrate the various features of C++ in real-life problems.

# Appendix A

## Project

### A.1 Menu Based Calculation System

#### Learning Objectives

The designing of the menu based calculation system project will help the students to:

- Create C++ classes with static functions
- Generate and call static functions
- Use the functions of Math.h header file
- Develop and display the main menu and its submenus

# Appendix H

## C++ Proficiency Test

### Part A

#### True / False Questions

State whether the following statements are true or false

1. A C++ program is identical to a C program with minor changes in coding.
2. Bundling functions and data together is known as data hiding.
3. In C++, a function contained within a class is called a member function.
4. Object modeling depicts the real-world entities more closely than do functions.
5. In using object-oriented languages like C++, we can define our own data types.
6. When a C++ program is executed, the function that appears first in the program is executed first.
7. In a 32-bit system, the data types **float** and **long** occupy the same number of bytes.
8. In an assignment statement such as

```
int x = expression;
```

the value of x is always equal to the value of the expression on the right.

9. In C++, declarations can appear almost anywhere in the body of a function.

## Proficiency Test

An appendix on proficiency test will help the readers assess their level of mastery on the language.

## Web Supplement

---

The following additional information is available on the web at <http://www.mhhe.com/balagurusamy/oop4e>

- ✱ Solution to the *Debugging Exercises*
- ✱ Chapter-wise *self-test quiz* with answers
- ✱ Complete code with step-by-step description and user manual for *Payroll Management Systems (Major Project)* and *Hospital Management Systems (Minor Project)*.
- ✱ Differences between ANSI C, C++ and ANSI/ISO C++

## Acknowledgements

---

Since the release of the first edition of this book a decade ago, lakhs of teachers, students and professional programmers have been using the book. Their overwhelming support encouraged me to bring out the Third Edition in 2006 and now the Fourth Edition.

My sincere thanks are due to the editorial and publishing professionals of Tata McGraw-Hill for their keen interest and support in bringing out this edition in the present form.

E BALAGURUSAMY



# Contents

*Preface*

*xiii*

## **1. Principles of Object-Oriented Programming 1**

- 1.1 Software Crisis 1
- 1.2 Software Evolution 3
- 1.3 A Look at Procedure-Oriented Programming 4
- 1.4 Object-Oriented Programming Paradigm 6
- 1.5 Basic Concepts of Object-Oriented Programming 7
- 1.6 Benefits of OOP 12
- 1.7 Object-Oriented Languages 13
- 1.8 Applications of OOP 14
- Summary* 15
- Review Questions* 17

## **2. Beginning with C++ 19**

- 2.1 What is C++? 19
- 2.2 Applications of C++ 20
- 2.3 A Simple C++ Program 20
- 2.4 More C++ Statements 25
- 2.5 An Example with Class 28
- 2.6 Structure of C++ Program 29
- 2.7 Creating the Source File 30
- 2.8 Compiling and Linking 30
- Summary* 31
- Review Questions* 32
- Debugging Exercises* 33
- Programming Exercises* 34

## **3. Tokens, Expressions and Control Structures 35**

- 3.1 Introduction 35
- 3.2 Tokens 36
- 3.3 Keywords 36
- 3.4 Identifiers and Constants 36
- 3.5 Basic Data Types 38
- 3.6 User-Defined Data Types 40
- 3.7 Derived Data Types 42

3.8	Symbolic Constants	43
3.9	Type Compatibility	45
3.10	Declaration of Variables	45
3.11	Dynamic Initialization of Variables	46
3.12	Reference Variables	47
3.13	Operators in C++	49
3.14	Scope Resolution Operator	50
3.15	Member Dereferencing Operators	52
3.16	Memory Management Operators	52
3.17	Manipulators	55
3.18	Type Cast Operator	57
3.19	Expressions and their Types	58
3.20	Special Assignment Expressions	60
3.21	Implicit Conversions	61
3.22	Operator Overloading	63
3.23	Operator Precedence	63
3.24	Control Structures	64
	<i>Summary</i>	69
	<i>Review Questions</i>	71
	<i>Debugging Exercises</i>	72
	<i>Programming Exercises</i>	75

---

## 4. Functions in C++

77

4.1	Introduction	77
4.2	The Main Function	78
4.3	Function Prototyping	79
4.4	Call by Reference	81
4.5	Return by Reference	82
4.6	Inline Functions	82
4.7	Default Arguments	84
4.8	const Arguments	87
4.9	Function Overloading	87
4.10	Friend and Virtual Functions	89
4.11	Math Library Functions	90
	<i>Summary</i>	90
	<i>Review Questions</i>	92
	<i>Debugging Exercises</i>	93
	<i>Programming Exercises</i>	95

---

## 5. Classes and Objects

96

5.1	Introduction	96
5.2	C Structures Revisited	97
5.3	Specifying a Class	99

5.4	Defining Member Functions	103
5.5	A C++ Program with Class	104
5.6	Making an Outside Function Inline	106
5.7	Nesting of Member Functions	107
5.8	Private Member Functions	108
5.9	Arrays within a Class	109
5.10	Memory Allocation for Objects	114
5.11	Static Data Members	115
5.12	Static Member Functions	117
5.13	Arrays of Objects	119
5.14	Objects as Function Arguments	122
5.15	Friendly Functions	124
5.16	Returning Objects	130
5.17	const Member Functions	132
5.18	Pointers to Members	132
5.19	Local Classes	134
	<i>Summary</i>	135
	<i>Review Questions</i>	136
	<i>Debugging Exercises</i>	137
	<i>Programming Exercises</i>	142

## 6. Constructors and Destructors

144

6.1	Introduction	144
6.2	Constructors	145
6.3	Parameterized Constructors	146
6.4	Multiple Constructors in a Class	150
6.5	Constructors with Default Arguments	153
6.6	Dynamic Initialization of Objects	153
6.7	Copy Constructor	156
6.8	Dynamic Constructors	158
6.9	Constructing Two-dimensional Arrays	160
6.10	const Objects	162
6.11	Destructors	162
	<i>Summary</i>	164
	<i>Review Questions</i>	165
	<i>Debugging Exercises</i>	166
	<i>Programming Exercises</i>	169

## 7. Operator Overloading and Type Conversions

171

7.1	Introduction	171
7.2	Defining Operator Overloading	172
7.3	Overloading Unary Operators	173
7.4	Overloading Binary Operators	176

- 7.5 Overloading Binary Operators Using Friends 179
- 7.6 Manipulation of Strings Using Operators 183
- 7.7 Rules for Overloading Operators 186
- 7.8 Type Conversions 187
  - Summary* 195
  - Review Questions* 196
  - Debugging Exercises* 197
  - Programming Exercises* 200

## **8. Inheritance: Extending Classes**

**201**

- 8.1 Introduction 201
- 8.2 Defining Derived Classes 202
- 8.3 Single Inheritance 204
- 8.4 Making a Private Member Inheritable 210
- 8.5 Multilevel Inheritance 213
- 8.6 Multiple Inheritance 218
- 8.7 Hierarchical Inheritance 224
- 8.8 Hybrid Inheritance 225
- 8.9 Virtual Base Classes 228
- 8.10 Abstract Classes 232
- 8.11 Constructors in Derived Classes 232
- 8.12 Member Classes: Nesting of Classes 240
  - Summary* 241
  - Review Questions* 243
  - Debugging Exercises* 243
  - Programming Exercises* 248

## **9. Pointers, Virtual Functions and Polymorphism**

**251**

- 9.1 Introduction 251
- 9.2 Pointers 253
- 9.3 Pointers to Objects 265
- 9.4 this Pointer 270
- 9.5 Pointers to Derived Classes 273
- 9.6 Virtual Functions 275
- 9.7 Pure Virtual Functions 281
  - Summary* 282
  - Review Questions* 283
  - Debugging Exercises* 284
  - Programming Exercises* 289

## **10. Managing Console I/O Operations**

**290**

- 10.1 Introduction 290
- 10.2 C++ Streams 291

- 10.3 C++ Stream Classes 292
- 10.4 Unformatted I/O Operations 292
- 10.5 Formatted Console I/O Operations 301
- 10.6 Managing Output with Manipulators 312
  - Summary* 317
  - Review Questions* 319
  - Debugging Exercises* 320
  - Programming Exercises* 321

## 11. Working with Files

323

- 11.1 Introduction 323
- 11.2 Classes for File Stream Operations 325
- 11.3 Opening and Closing a File 325
- 11.4 Detecting end-of-file 334
- 11.5 More about Open(): File Modes 334
- 11.6 File Pointers and Their Manipulations 335
- 11.7 Sequential Input and Output Operations 338
- 11.8 Updating a File: Random Access 343
- 11.9 Error Handling During File Operations 348
- 11.10 Command-line Arguments 350
  - Summary* 353
  - Review Questions* 355
  - Debugging Exercises* 356
  - Programming Exercises* 358

## 12. Templates

359

- 12.1 Introduction 359
- 12.2 Class Templates 360
- 12.3 Class Templates with Multiple Parameters 365
- 12.4 Function Templates 366
- 12.5 Function Templates with Multiple Parameters 371
- 12.6 Overloading of Template Functions 372
- 12.7 Member Function Templates 373
- 12.8 Non-Type Template Arguments 374
  - Summary* 375
  - Review Questions* 376
  - Debugging Exercises* 377
  - Programming Exercises* 379

## 13. Exception Handling

380

- 13.1 Introduction 380
- 13.2 Basics of Exception Handling 381

- 13.3 Exception Handling Mechanism 381
- 13.4 Throwing Mechanism 386
- 13.5 Catching Mechanism 386
- 13.6 Rethrowing an Exception 391
- 13.7 Specifying Exceptions 392
  - Summary* 394
  - Review Questions* 395
  - Debugging Exercises* 396
  - Programming Exercises* 400

---

## **14. Introduction to the Standard Template Library**

**401**

- 14.1 Introduction 401
- 14.2 Components of STL 402
- 14.3 Containers 403
- 14.4 Algorithms 406
- 14.5 Iterators 408
- 14.6 Application of Container Classes 409
- 14.7 Function Objects 419
  - Summary* 421
  - Review Questions* 423
  - Debugging Exercises* 424
  - Programming Exercises* 426

---

## **15. Manipulating Strings**

**428**

- 15.1 Introduction 428
- 15.2 Creating (string) Objects 430
- 15.3 Manipulating String Objects 432
- 15.4 Relational Operations 433
- 15.5 String Characteristics 434
- 15.6 Accessing Characters in Strings 436
- 15.7 Comparing and Swapping 438
  - Summary* 440
  - Review Questions* 441
  - Debugging Exercises* 442
  - Programming Exercises* 445

---

## **16. New Features of ANSI C++ Standard**

**446**

- 16.1 Introduction 446
- 16.2 New Data Types 447
- 16.3 New Operators 449
- 16.4 Class Implementation 451

16.5	Namespace Scope	453
16.6	Operator Keywords	459
16.7	New Keywords	460
16.8	New Headers	461
	<i>Summary</i>	461
	<i>Review Questions</i>	463
	<i>Debugging Exercises</i>	464
	<i>Programming Exercises</i>	467

## **17. Object-Oriented Systems Development**

**468**

17.1	Introduction	468
17.2	Procedure-Oriented Paradigms	469
17.3	Procedure-Oriented Development Tools	472
17.4	Object-Oriented Paradigm	473
17.5	Object-Oriented Notations and Graphs	475
17.6	Steps in Object-Oriented Analysis	479
17.7	Steps in Object-Oriented Design	483
17.8	Implementation	490
17.9	Prototyping Paradigm	490
7.10	Wrapping Up	491
	<i>Summary</i>	492
	<i>Review Questions</i>	494

<b>Appendix A: Projects</b>	<b>496</b>
<b>Appendix B: Executing Turbo C++</b>	<b>539</b>
<b>Appendix C: Executing C++ Under Windows</b>	<b>552</b>
<b>Appendix D: Glossary of ANSI C++ Keywords</b>	<b>564</b>
<b>Appendix E: C++ Operator Precedence</b>	<b>570</b>
<b>Appendix F: Points to Remember</b>	<b>572</b>
<b>Appendix G: Glossary of Important C++ and OOP Terms</b>	<b>584</b>
<b>Appendix H: C++ Proficiency Test</b>	<b>596</b>
<b>Bibliography</b>	<b>632</b>

# 1

## Principles of Object-Oriented Programming

### Key Concepts

- Software evolution
- Procedure-oriented programming
- Object-oriented programming
- Objects
- Classes
- Data abstraction
- Encapsulation
- Inheritance
- Polymorphism
- Dynamic binding
- Message passing
- Object-oriented languages
- Object-based languages

### 1.1 Software Crisis

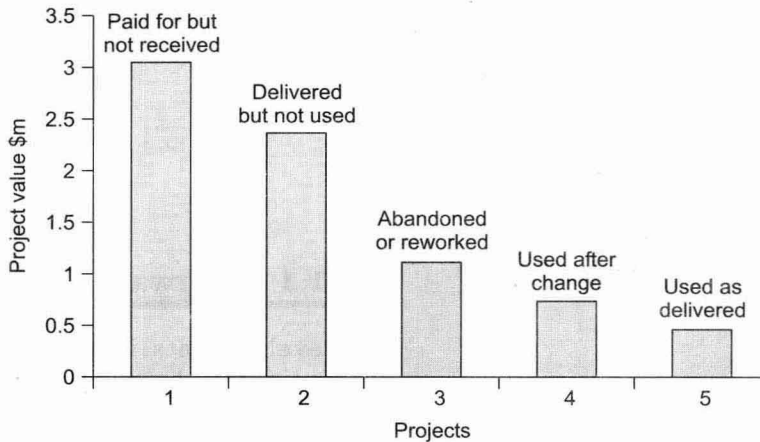
Developments in software technology continue to be dynamic. New tools and techniques are announced in quick succession. This has forced the software engineers and industry to continuously look for new approaches to software design and development, and they are becoming more and more critical in view of the increasing complexity of software systems as well as the highly competitive nature of the industry. These rapid advances appear to have created a situation of crisis within the industry. The following issues need to be addressed to face this crisis:

- How to represent real-life entities of problems in system design?
- How to design systems with open interfaces?



- ✱ How to ensure reusability and extensibility of modules?
- ✱ How to develop modules that are tolerant to any changes in future?
- ✱ How to improve software productivity and decrease software cost?
- ✱ How to improve the quality of software?
- ✱ How to manage time schedules?
- ✱ How to industrialize the software development process?

Many software products are either not finished, or not used, or else are delivered with major errors. Figure 1.1 shows the fate of the US defence software projects undertaken in the 1970s. Around 50% of the software products were never delivered, and one-third of those which were delivered were never used. It is interesting to note that only 2% were used as delivered, without being subjected to any changes. This illustrates that the software industry has a remarkably bad record in delivering products.



**Fig. 1.1** ⇔ *The state of US defence projects (according to the US government)*

Changes in user requirements have always been a major problem. Another study (Fig. 1.2) shows that more than 50% of the systems required modifications due to changes in user requirements and data formats. It only illustrates that, in a changing world with a dynamic business environment, requests for change are unavoidable and therefore systems must be adaptable and tolerant to changes.

These studies and other reports on software implementation suggest that software products should be evaluated carefully for their quality before they are delivered and implemented. Some of the quality issues that must be considered for critical evaluation are:

1. Correctness
2. Maintainability
3. Reusability
4. Openness and interoperability