




Estimating SOFTWARE COSTS

BRINGING REALISM TO ESTIMATING

Second Edition

- 
- Complete software projects on time and within budget
 - Select the estimation applications and methods that fit your needs
 - Use object-oriented, reusable component, and Java techniques
 - Understand impact of Agile, Extreme (XP), and CMM on schedules and costs

CAPERS JONES

FOREWORD BY Doug Brindley, President, Software Productivity Research, LLC

Estimating Software Costs

ABOUT THE AUTHOR

CAPERS JONES is a leading authority in the world of software estimating, measurement, metrics, productivity, and quality. He designed IBM's first automated software estimating tool in 1973. He also designed both the SPQR20[®] and Checkpoint[®] commercial software cost-estimating tools. He was the founder and chairman of Software Productivity Research (SPR), where he retains the title of Chief Scientist Emeritus. While he was chairman of SPR, the company brought out the leading KnowledgePlan[®] estimating tool under his direction. Jones frequently speaks at conferences such as IEEE Software, International Function Point Users Group (IFPUG), the Project Management Institute (PMI), the Software Process Improvement Network (SPIN), the Computer Aid (CAI) conference series, and at scores of in-house corporate and government events. Capers Jones has been named to the Computer Aid advisory board in 2007. *Estimating Software Costs* is his 14th book.

Estimating Software Costs: Bringing Realism to Estimating, Second Edition

Capers Jones



New York Chicago San Francisco Lisbon London Madrid
Mexico City Milan New Delhi San Juan Seoul
Singapore Sydney Toronto

Copyright © 2007 by The McGraw-Hill Companies. All rights reserved. Except as permitted under the United States Copyright Act of 1976, no part of this publication may be reproduced or distributed in any form or by any means, or stored in a database or retrieval system, without the prior written permission of the publisher.

ISBN: 978-0-07-177679-0

MHID: 0-07-177679-6

The material in this eBook also appears in the print version of this title: ISBN: 978-0-07-148300-1, MHID: 0-07-148300-4.

All trademarks are trademarks of their respective owners. Rather than put a trademark symbol after every occurrence of a trademarked name, we use names in an editorial fashion only, and to the benefit of the trademark owner, with no intention of infringement of the trademark. Where such designations appear in this book, they have been printed with initial caps.

McGraw-Hill eBooks are available at special quantity discounts to use as premiums and sales promotions, or for use in corporate training programs. To contact a representative please e-mail us at bulksales@mcgraw-hill.com.

Information has been obtained by McGraw-Hill from sources believed to be reliable. However, because of the possibility of human or mechanical error by our sources, McGraw-Hill, or others, McGraw-Hill does not guarantee the accuracy, adequacy, or completeness of any information and is not responsible for any errors or omissions or the results obtained from the use of such information.

TERMS OF USE

This is a copyrighted work and The McGraw-Hill Companies, Inc. ("McGrawHill") and its licensors reserve all rights in and to the work. Use of this work is subject to these terms. Except as permitted under the Copyright Act of 1976 and the right to store and retrieve one copy of the work, you may not decompile, disassemble, reverse engineer, reproduce, modify, create derivative works based upon, transmit, distribute, disseminate, sell, publish or sublicense the work or any part of it without McGraw-Hill's prior consent. You may use the work for your own noncommercial and personal use; any other use of the work is strictly prohibited. Your right to use the work may be terminated if you fail to comply with these terms.

THE WORK IS PROVIDED "AS IS." McGRAW-HILL AND ITS LICENSORS MAKE NO GUARANTEES OR WARRANTIES AS TO THE ACCURACY, ADEQUACY OR COMPLETENESS OF OR RESULTS TO BE OBTAINED FROM USING THE WORK, INCLUDING ANY INFORMATION THAT CAN BE ACCESSED THROUGH THE WORK VIA HYPERLINK OR OTHERWISE, AND EXPRESSLY DISCLAIM ANY WARRANTY, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. McGraw-Hill and its licensors do not warrant or guarantee that the functions contained in the work will meet your requirements or that its operation will be uninterrupted or error free. Neither McGraw-Hill nor its licensors shall be liable to you or anyone else for any inaccuracy, error or omission, regardless of cause, in the work or for any damages resulting therefrom. McGraw-Hill has no responsibility for the content of any information accessed through the work. Under no circumstances shall McGraw-Hill and/or its licensors be liable for any indirect, incidental, special, punitive, consequential or similar damages that result from the use of or inability to use the work, even if any of them has been advised of the possibility of such damages. This limitation of liability shall apply to any claim or cause whatsoever whether such claim or cause arises in contract, tort or otherwise.

*This book is dedicated to colleagues
who were among the pioneers of software
measurement and software cost estimating:
Al Albrecht, Dr. Barry Boehm, Tom DeMarco,
Steve Kan, Larry Putnam, Howard Rubin,
and Tony Salvaggio*

This page intentionally left blank

Contents

Foreword	xv
Preface	xix
Acknowledgments	xxv

SECTION 1 INTRODUCTION TO SOFTWARE COST ESTIMATION

Chapter 1. Introduction	3
How Software Cost-Estimating Tools Work	4
Cautions About Accidental Omissions from Estimates	15
Software Cost Estimating and Other Development Activities	17
References	20
Chapter 2. The Origins of Software Cost Estimation	23
The Early History of Software Cost Estimation	24
The Expansion and Use of Functional Metrics for Software Cost Estimating	28
References	32
Chapter 3. Six Forms of Software Cost Estimation	33
Overview of Manual Software-Estimating Methods	34
Overview of Automated Software-Estimating Methods	36
Comparison of Manual and Automated Estimates for Large Software Projects	48
References	49
Chapter 4. Software Cost-Estimating Tools and Project Success and Failure Rates	53
Probabilities of Software Project Success or Failure	55
References	59

Chapter 5. Sources of Error in Software Cost Estimation	61
Judging the Accuracy of Software Cost Estimates	65
Classes of Software Estimation Errors	68
References	86
 SECTION 2 PRELIMINARY ESTIMATION METHODS	
 Chapter 6. Manual Software-Estimating Methods	91
Rules of Thumb Based on Lines-of-Code Metrics	92
Rules of Thumb Based on Ratios and Percentages	95
Rules of Thumb Based on Function Point Metrics	99
Function Point Sizing Rules of Thumb	102
Rules of Thumb for Schedules, Resources, and Costs	117
Rules of Thumb Using Activity-Based Cost Analysis	121
Summary and Conclusions	127
References	128
 Chapter 7. Manual Estimating Methods Derived from Agile Projects and New Environments	131
Metrics Used for Rules of Thumb	135
Rules of Thumb for Manual Software Cost Estimates	140
Component-Based Development	143
Dynamic System Development Method (DSDM)	146
Enterprise Resource Planning (ERP) Deployment	148
Extreme Programming (XP)	152
International Outsourcing	155
Object-Oriented (OO) Development	159
Capability Maturity Model (CMM)	162
Software Methods with Only Partial Rules of Thumb	167
Cleanroom Development	167
Crystal Development Approach	168
Feature-Driven Development (FDD)	168
ISO 9000-9004 Quality Standards	169
Iterative or Incremental Development	169
Pattern-Based Software Development	171
Quality Function Deployment (QFD)	174
Rapid Application Development (RAD)	175
Scrum	176
six-sigma for Software	177
Spiral Software Development	179
Unified Modeling Language (UML)	180
Use Cases for Software Requirements	181
Web-Based Applications	183
Summary and Conclusions	185
References	185

Chapter 8. Automated Estimates from Minimal Data	189
Stage 1: Recording Administrative and Project Information	190
Stage 2: Preliminary Sizing of Key Deliverables	203
Stage 3: Producing a Preliminary Cost Estimate	219
Summary and Conclusions	224
References	225

SECTION 3 SIZING SOFTWARE DELIVERABLES

Chapter 9. Sizing Software Deliverables	229
General Sizing Logic for Key Deliverables	229
Sizing Methods Circa 2007	230
Pattern Matching from Historical Data	232
Using Historical Data to Predict Growth in Requirements	233
Mathematical or Statistical Attempts to Extrapolate Size from Partial Requirements	234
Arbitrary Rules of Thumb for Adding Contingency Factors	235
Freezing Requirements at Fixed Points in Time	236
Producing Formal Cost Estimates Only for Subsets of the Total Application	237
Volume of Function Point Data Available	245
Software Complexity Analysis	247
Software Sizing with Reusable Components	258
Overview of the Basic Forms of Software Sizing Metrics	260
Source Code Sizing	269
Sizing Object-Oriented Software Projects	275
Sizing Text-Based Paper Documents	277
Sizing Graphics and Illustrations	283
Sizing Bugs or Defects	286
Sizing Test Cases	293
The Event Horizon for Sizing Software Artifacts	295
What Is Known as a Result of Sizing Software Projects	297
Strengths and Weaknesses of Software Size Metrics	299
Summary and Conclusions	301
References	302

SECTION 4 COST-ESTIMATING ADJUSTMENT FACTORS

Chapter 10. Compensation and Work-Pattern Adjustments	307
Manual and Automated Methods of Adjustment	308
Exclusions from Normal Software Cost Estimates	310
Setting Up the Initial Conditions for a Cost Estimate	313
Variations in Burden Rates or Overhead Costs	316
Variations in Work Habits and Unpaid Overtime	318
References	324

Chapter 11. Activity Pattern Adjustment Factors	325
Twenty Five Common Activities for Software Projects	326
References	332
Chapter 12. Software Technology Adjustment Factors	335
Adjustment Factors and Macro-Estimation Tools	336
Factors That Influence Software Development Productivity	340
Factors That Influence Software Maintenance Productivity	343
Patterns of Positive and Negative Factors	345
Adjustment Factors and Micro-Estimating Tools	347
References	362
SECTION 5 ACTIVITY-BASED SOFTWARE COST ESTIMATING	
Chapter 13. Estimating Software Requirements	367
Function Points and Software Requirements	374
Primary Topics for Software Requirements	381
Secondary Topics for Software Requirements	382
Positive and Negative Requirements Adjustment Factors	383
Requirements and End-User Software	386
Requirements and Agile Applications	386
Requirements and Management Information Systems (MIS) Projects	386
Requirements and Outsourced Projects	387
Requirements and Systems Software	387
Requirements and Commercial Software	388
Requirements and Military Software Projects	389
Requirements and Web-Based Applications	390
Evaluating Combinations of Requirements Factors	390
References	393
Chapter 14. Estimating Software Prototypes	395
Disposable Prototypes	398
Time box Prototypes	398
Evolutionary Prototypes	400
Default Values for Estimating Disposable Prototypes	402
Positive and Negative Factors That Influence Software Prototypes	404
References	407
Chapter 15. Estimating Software Specifications and Design	409
Positive Design Adjustment Factors	414
Negative Design Adjustment Factors	415
References	418

Chapter 16. Estimating Design Inspections	421
Inspection Literature	421
Inspection Process	423
Value of Inspections	426
References	432
 Chapter 17. Estimating Programming or Coding	 435
The Impact of Reusability on Programming	442
The Impact of Experience on Programming	444
The Impact of Bugs or Errors on Programming	444
The Impact of Unpaid Overtime on Programming	446
The Impact of Creeping Requirements on Programming	448
The Impact of Code Structure and Complexity on Programming	449
The Impact of Unplanned Interruptions on Programming	450
The Impact of Application Size on Programming	451
The Impact of Office Space and Ergonomics on Programming	452
The Impact of Tools on Programming	454
The Impact of Programming Languages on Programming	455
The Impact of Schedule Pressure on Programming	459
References	459
 Chapter 18. Estimating Code Inspections	 461
Code Inspection Literature	461
Effectiveness of Code Inspections	462
Considerations for Estimating Code Inspections	466
References	470
 Chapter 19. Estimating Software Configuration Control and Change Management	 471
The Literature on Change Management	473
Measuring Software Change	475
Changes in User Requirements	479
Changes in Specifications and Design	480
Changes Due to Bugs or Defect Reports	481
Summary and Conclusions	482
References	483
 Chapter 20. Estimating Software Testing	 485
General Forms of Software Testing	491
Specialized Forms of Software Testing	495
Forms of Testing Involving Users or Clients	498
Number of Testing Stages	499
Testing Pattern Variations by Industry and Type of Software	501
Testing Pattern Variations by Size of Application	503

Testing Stages Noted in Lawsuits Alleging Poor Quality	504
Using Function Points to Estimate Test-Case Volumes	505
Using Function Points to Estimate the Numbers of Test Personnel	507
Testing and Defect-Removal Efficiency Levels	508
Using Function Points to Estimate Testing Effort and Costs	510
Testing by Developers or by Professional Test Personnel	512
Test Case Coverage	514
The Factors That Affect Testing Performance	514
References	515

Chapter 21. Estimating User and Project Documentation 519

Estimating Tools and Software Documentation	521
Quantifying the Number and Sizes of Software Document Types	523
Software Documentation Tools on Lagging and Leading Projects	527
References	529

Chapter 22. Estimating Software Project Management 531

The Roles of Software Project Management	535
Project Managers Who Are Also Technical Contributors	537
Project Management for Hybrid Projects Involving Hardware and Software	538
Project Management and External Schedule Pressures	538
Project Management Tools	539
Project Management on Large Systems with Many Managers	542
Time-Splitting, or Managing Several Projects Simultaneously	544
The Span of Control, or Number of Staff Members per Manager	545
Managing Multiple Occupation Groups	546
The Presence or Absence of Project Offices for Large Systems	548
Experience Levels of Software Project Managers	549
Quality-Control Methods Selected by Project Managers	550
Project Managers and Metrics	551
Summary of Project Management Findings	551
References	551

SECTION 6 MAINTENANCE AND ENHANCEMENT COST ESTIMATING

Chapter 23. Maintenance and Enhancement Estimating 557

Nominal Default Values for Maintenance and Enhancement Activities	562
Metrics and Measurement Problems with Small Maintenance Projects	566
Best and Worst Practices in Software Maintenance	567
Software Entropy and Total Cost of Ownership	570
Installing New Releases and Patches from Software Vendors	572
Major Enhancements	573
Minor Enhancements	574

Maintenance (Defect Repairs)	576
Warranty Repairs	581
Customer Support	581
Economics of Error-Prone Modules	582
Mandatory Changes	584
Complexity Analysis	585
Code Restructuring and Refactoring	586
Performance Optimization	588
Migration Across Platforms	588
Conversion to New Architectures	589
Reverse Engineering	589
Re-engineering	590
Dead Code Removal	590
Dormant Application Removal	591
Nationalization	591
Mass Update Projects	592
Retirement or Withdrawal of Applications	593
Field Service	594
Combinations and Concurrent Maintenance Operations	594
References	599

Chapter 24. Software Cost-Estimating Research Issues 603

Metrics Conversion	604
Automatic Sizing from User Requirements	606
Activity-Based Costs for Agile, Object-Oriented, and Web Projects	608
Complexity Analysis of Software Applications	610
Value Analysis of Software Applications	611
Risk Analysis and Software Cost Estimates	613
Including Specialists in Software Cost Estimates	615
Reuse Analysis and Software Cost Estimates	617
Process Improvement Estimation	622
Methodology Analysis and Software Cost Estimating	625
Summary and Conclusions About Software Cost-Estimating Research	629

Index 631

This page intentionally left blank

Foreword

Once when discussing a project then already months behind schedule, a puzzled executive observed that “every project we do comes in late.” Turning to his cowering CIO, he further fumed, “We give you reasonable deadlines *up front!* Why can’t you manage to them?” A knowing look passed between the CIO and a certain consultant who shall remain nameless. Setting the date before the requirements are set is the oldest problem in the software book, but the CIO had no credibility when he argued for different dates, since he lacked the one thing he needed to command the respect of his bosses: a well-defined, consistently applied, and rigorously executed software estimation process.

In its first fifty years, the business of software development acquired a notorious reputation for out-of-control schedules, massive cost overruns, and imperceptible quality control. Project estimation, planning, and quality management were frequently so primitive and haphazard that most large software development projects ran late and exceeded their original budgets. Indeed, many such projects were canceled before they ever reached completion, typically after wasting vast quantities of human and financial resources.

Of course, the worst cases of project failure—the really big fiascos—were often covered up by organizations, since public knowledge of the worst failures would severely impact the market value and public perception of otherwise successful companies.

Naturally, each project had its own story, but a common thread seemed to run through the worst cases. Most overruns and failures to deliver were based on careless, arbitrary, and/or grossly optimistic cost, effort and duration estimates that were typically created with manual, “seat of the pants” estimation techniques. From the beginning of software cost estimation, the “seat of the pants” approach was immensely popular, and as the comic in an old burlesque routine used to say, “It’s been following us around ever since.”

Early on, it was fairly easy to estimate the anticipated size of software applications, since the few programming languages were very similar

and closely resembled actual machine instructions. In this sense, form and function were, if only for a brief time, very nearly equal. Similarly, individual programmer productivity was reasonably easy to predict, expressed usually as “lines of code per month.” Thus, a project manager could build a “back-of-the-envelope” estimate by dividing total estimated size by the average productivity, yielding an estimate of the number of months needed to write the code.

Today, when coding can consume as little as 5 percent of a software project’s lifecycle, such primitive estimation techniques seem almost comical. Nevertheless, in many organizations the process works about the same way it did 50 years ago. In reality, “back of the envelope” is just a nicer way of saying “seat of the pants,” and from an even casual survey of cost and schedule overruns, one might say that the “seat of the pants” is wearing a little thin.

It would be a great relief to say that all that is changing in these latter days of enlightenment, marked by whiz-bang development tools, off-the-shelf enterprise applications that do it all right out of the box (well, almost), black-belted six-sigma samurais, and process improvement gurus with their amply documented how-to’s and must-do’s. It ought to be getting so much better now. However, if one were to guess, it’s likely that 50 years from *now* it will still be said that in its infancy—and now in its presumed adolescence—the software industry was most notable for its stubborn and almost entirely successful resistance to anything like standard engineering disciplines, including in particular rigorous, repeatable calibration, measurement, and estimation.

Thankfully, there have been brave voices in the software wilderness over the years, who studied and sought to remedy the basic problem of poor software project estimation. Capers Jones, Founder and Chief Scientist of Software Productivity Research, LLC (SPR), is perhaps the most widely published and read of this small group of passionate pioneers. From his early work in linguistics and programming language dynamics to his development of a highly successful line of commercial parametric estimating models, including today’s SPR KnowledgePlan[®], Jones has been a consistent advocate for a more formalized software cost-estimation process.

In 1998, Jones published the first edition of *Estimating Software Costs*, which comprehensively surveyed the history of software estimation, as well as the full range of estimation tools and techniques available to project managers at the time. With this current revised second edition, he expands the scope of his study to include observations and commentary on the current state of estimation in numerous new outposts on the software development landscape, including extreme programming, Agile methods, and ever more extensive ERP and COTS solutions.

In addition, Jones provides the latest data from SPR's research on the myriad of factors affecting project estimation.

Jones continues to argue that organizations must abandon the concept of software estimation as a linear application of industrial principles. His central thesis is that successful software development is not a simple matter of matching a number of people to a number of work units in order to achieve arbitrary budgets and deadlines—an approach that continues to lead to the “endless project” phenomenon, otherwise known as the “death march,” the “train wreck,” or the “black hole.” (Frequently, this approach also leads to a new CIO.) He reminds us that the software process itself remains exceptionally people-oriented and so is somewhat resistant to the influence of newer and better technologies. He points out that ever-advancing tools and methodologies have indeed tended to improve productivity over time, but costs have not always declined proportionately, and the people on a project—their skills, their perceptions, even their feelings—can still matter the most. Somehow, estimates must take these into account.

The overarching problem with poor estimation continues to be excessive optimism, brought on largely by poor methods and arbitrary expectations. However, most software cost estimation is still done by individual project managers working with primitive, homegrown methods. Some are successful, but the ubiquity of cost overruns, schedule slippages, and canceled projects across every industry suggests that most are not. Something needs to change, and Capers Jones effectively argues that formal, rigorous estimation methods—and institutional commitment to use them—are the necessary agents of this change.

—Doug Brindley
President, Software Productivity Research LLC

This page intentionally left blank

Preface

In the ten years between the first and the second editions of this book, many changes have taken place in the computer industry and also in the technology of software cost estimating.

More than 20 new development methodologies have appeared, including about a dozen kinds of “Agile” development. Object-oriented (OO) methods are expanding in popularity. Use cases and the unified modeling language (UML) have joined the mainstream of software design methods. The Software Engineering Institute has issued the new capability maturity model integration (CMMI). All of these new approaches are being used on software projects where accurate cost and schedule estimates are important.

Some of these new methods have created new metrics for estimation and measurement. Thus in 2007 software project managers may have to understand not only lines of code and function point metrics, but also Cosmic function points, engineering function points, object points, story points, web object points, use case points, and perhaps 30 others. However, these new metrics are still experimental and lack large volumes of historical data. Standard function point metrics have been used to measure more than 25,000 software projects. So far as can be determined, all of the other metrics put together have been used to measure less than 300 projects, or perhaps 10 projects per metric.

There has also been significant research into estimating methods, and also research into the reasons for inaccurate estimation. While estimating errors are still common, we now know the primary causes of such errors. There are four of them:

- Software projects grow at a rate of about 2 percent per month during development.
- Excessive numbers of bugs or defects throw off testing schedules.
- Producing paper documents often costs more than source code.
- Accurate estimates may be arbitrarily replaced by optimistic estimates.

Now that we know the major sources of software cost-estimating errors, we can begin to control them. The three main technical sources of software cost-estimating errors are actually capable of being eliminated from software cost estimates. But the fourth source of error, the arbitrary replacement of accurate estimates with optimistic ones, remains troublesome.

Once created, software cost estimates must be approved by the clients for whom the software project is being created and sometimes by higher management as well. When clients or higher management face urgent business needs, there is a strong tendency for them to reject accurate cost estimates. This is because building large or complex software applications remains time consuming and expensive.

Thus, instead of starting a software project with a true cost estimate based on team and technical capabilities, many software projects are forced to use external business dates as deadlines. They are also forced to use budgets that are lower than necessary to include all needed functions. These are sociological issues and they are difficult to solve.

The best solution to avoid replacement of accurate estimates is to support the estimate with historical data from similar projects. If history proves that a specific kind of project has never been built any faster or cheaper than the formal cost estimate, then the estimate might survive. Without solid historical data, the estimate will probably be rejected and replaced. This means that benchmarking, or the collection of accurate historical data, is an important precursor to accurate estimation.

Collecting historical data and designing and building software cost-estimating tools have been my main occupations since 1971. My estimation work started at IBM when a colleague and I were asked to collect data about the factors that affected software costs.

After spending a year on collecting cost data within IBM and reviewing the external literature on software costs, it seemed possible to construct a rule-based automated estimating tool that could predict software effort, schedules, and costs for the main activities of IBM's systems software development projects; that is, requirements, design, design reviews, coding, code inspections, testing, user documentation, and project management.

A proposal was given to IBM senior management to fund the development of a software cost-estimating tool. The proposal was accepted, and work commenced in 1972. From that point on, I've designed and built a dozen software cost-estimating tools for individual companies or for the commercial estimating market. The purpose of this book is to show some of the kinds of information needed to build software cost models, and to present an insider's view of the cost-estimating business.

The book tries to cover the fundamental issues involved with software cost estimating. A number of other estimating and metrics specialists have contributed ideas and information and are cited many times.

The work of such other estimating specialists as Allan Albrecht, Dr. Barry Boehm, Frank Freiman, Don Galorath, Dr. Randall Jensen, Steve McConnell, Larry Putnam, Dr. Howard Rubin, and Charles Symons are discussed, although as competitors we do not often share proprietary information with each other.

In my view and also in the view of my competitors, accurate software cost estimating is important to the global economy. Every software project manager, every software quality assurance specialist, and many software engineers, systems analysts, and programmers should understand the basic concepts of software cost estimation. This is a view shared by all of the commercial software cost-estimating vendors.

All of us in the commercial software cost-estimating business know dozens of manual estimating algorithms. All of us in the cost-estimating business use manual estimating methods from time to time on small projects, but not one of us regards manual estimation methods as being sufficient for handling the full life-cycle estimates of major software projects. If manual methods sufficed, there would be no commercial software estimating tools.

It is an interesting phenomenon that most of the major overruns and software disasters are built upon careless and grossly optimistic cost estimates, usually done manually. Projects that use formal estimating tools, such as the competitive tools COCOMO II, GECOMO, SLIM, PRICE-S, ProQMS, SEER, SoftCost, or my company's tools (SPQR/20, CHECKPOINT, or KnowledgePlan), usually have much better track records of staying within their budgets and actually finishing the project without serious mishap.

The reason that manual estimating methods fail for large systems can be expressed in a single word: *complexity*. There are hundreds of factors that determine the outcome of a software project, and it is not possible to deal with the combinations of these factors using simple algorithms and manual methods.

This book illustrates the kinds of complex estimating problems that triggered the creation of the cost-estimating and software project management tool industries. The problems of large-system estimation are the main topic of concern, although estimating methods for small projects are discussed, too.

Estimation is not the only project management function that now has automated support. The book also attempts to place the subject of estimation tools in context with other kinds of project management tools, and to point out gaps where additional kinds of tools are needed.

Software cost-estimating tools are now part of a suite of software project management tools that includes cost estimating, quality estimating, schedule planning (often termed *project management*), methodology or

process management, risk analysis, departmental budgeting, milestone tracking, cost reporting, and variance analysis.

These disparate tools have not yet reached the level of sophisticated and seamless integration that has been achieved in the domain of word processing coupled with spreadsheets, databases, and graphical tools, but that level of integration is on the horizon.

Because software cost estimating is a very complex activity involving scores of factors and hundreds of adjustments, this book has a fairly complex structure. It is divided into six main sections and 24 individual chapters.

Section 1 includes an introduction to the topic of software cost estimation, and a survey of the features of software cost-estimating tools. Section 1 also covers the business aspects of software project management, such as how much tools are likely to cost and what kind of value they create. Section 1 assumes no prior knowledge on the part of the reader.

Section 2 deals with several methods for creating early estimates, long before a project's requirements are completely understood. Early estimation from partial knowledge is one of the most difficult forms of estimation, and yet one of the most important. Far too often, early estimates end up being "engraved in stone" or becoming the official estimate for a software project.

This section discusses both manual estimating methods using rules of thumb and the somewhat more sophisticated preliminary estimating methods offered by commercial software estimating tools.

Section 3 deals with the methods of sizing various software artifacts. All commercial software estimating tools need some form of size information in order to operate, and there are a surprisingly large number of ways for dealing with size.

As this book is being written, sizing based on function point metrics is dominant in the software estimating world, but sizing based on lines of code, and on less tangible materials, also occurs. There are also some new experimental metrics such as "use case points" and "object points." However, these are primarily experimental and lack significant amounts of historical data. There are also a number of specialized metrics in the object-oriented domain. While interesting and useful for OO projects, the specialized OO metrics cannot be used for side-by-side comparisons between OO projects and conventional projects.

Section 4 deals with how software cost-estimating tools handle adjustment factors. There are more than 100 known factors that can influence the outcomes of software projects, including the capabilities of the team, the presence or absence of overtime, the methodologies and tools used, and even office space and ergonomics.

Although commercial software cost-estimating tools offer default values for many important topics, the ranges of uncertainty are so great

that users are well advised to replace generic “industry averages” with specific values from their own enterprises for key parameters, such as average salary levels, burden rates, staff experience levels, and other factors that can exert a major impact on final results.

Section 5 deals with the principles of activity-based cost estimation for ten activities that occur with high frequency on many software projects:

- Requirements gathering and analysis
- Prototyping
- Specifications and design
- Formal design inspections
- Coding
- Formal code inspections
- Change management, or configuration control
- User documentation
- Testing
- Project management

There are, of course many more than ten activities, but these ten were selected because they occur with high frequency on a great many software projects. Unless the estimates for these ten activities are accurate, there is little hope for accuracy at the gross project level.

Section 6 deals with the principles of activity-based cost estimating for 21 kinds of maintenance and enhancement activities. Maintenance estimating is far more complex than development estimating, since the age and structure of the base application has a severe impact.

There are also a number of very specialized kinds of maintenance estimating that can be quite expensive when they occur, but do not occur with high frequency—error-prone module removal, field service, and dealing with *abeyant* defects, which occur when a user runs an application but cannot be replicated at the maintenance repair facility. Section 6 also deals with several new forms of maintenance that are extremely costly—special dates, expansion of the number of digits in phone numbers, changing daylight savings time, and similar problems that affect hundreds of applications.

– *Capers Jones*

This page intentionally left blank

Acknowledgments

As always, thanks to my wife, Eileen Jones, for making this book possible in many ways. She handles all of our publishing contracts, and by now knows the details of these contracts as well as some attorneys. Thanks also for her patience when I get involved in writing and disappear into our computer room. Also thanks for her patience on holidays and vacations when I take my portable computer.

Appreciation is due to Michael Bragen, Doug Brindley, and Peter Katsoulas of Software Productivity Research LLC. Doug, Michael, and Peter are continuing to develop and refine tools based on some of my estimating algorithms. Thanks to Chas Dougliis for his help over many years.

Appreciation is also due to Charles Dougliis and Mark Pinis for their design of SPR's KnowledgePlan estimating tool.

Thanks to Tony Salvaggio and Michael Milutis of ComputerAid for providing many opportunities to discuss estimation and benchmarking with their clients and their colleagues. Thanks also for their interest in publishing my reports on their web site.

Appreciation is due to hundreds of colleagues who are members of the International Function Point Users Group (IFPUG). Without IFPUG and the ever-expanding set of projects measured using function points, estimating technology would not have reached the current level of performance.

Thanks are due to Dave Shough, a colleague at IBM who helped me collect data for my first estimating tool. Thanks, too, to Dr. Charles Turk, another IBM colleague and a world-class APL expert, who built the first estimating tool that I designed. Appreciation is due to the late Ted Climis of IBM, who sponsored much of my research on software costs.

Special appreciation is due to Jim Frame, who passed away in October 1997 just as the first edition of this manuscript was being finished. Jim was the Director of IBM's Languages and Data Facilities laboratories, and the immediate sponsor of my first studies on software cost estimation. Jim was later ITT's vice president of programming, where he also

supported software cost-estimation research. The software industry lost a leading figure with his passing. Jim's vision of the important role software plays in modern business inspired all who knew him.

Great appreciation is due to all of my former colleagues at Software Productivity Research for their aid in gathering the data, assisting our clients, building the tools that we use, and for making SPR an enjoyable environment. Special thanks to the families and friends of the SPR staff, who had to put up with lots of travel and far too much overtime. Thanks to my former colleagues Mark Beckley, Ed Begley, Chuck Berlin, Barbara Bloom, Julie Bonaiuto, William Bowen, Michael Bragen, Doug Brindley, Kristin Brooks, Tom Cagley, Lynne Caramanica, Debbie Chapman, Sudip Charkraboty, Craig Chamberlin, Carol Chiungos, Michael Cunnane, Chas Dougliis, Charlie Duczakowski, Gail Flaherty, Dave Garmus, Richard Gazoorian, James Glorie, Scott Goldfarb, Jane Green, David Gustafson, Wayne Hadlock, Bill Harmon, Shane Hartman, Bob Haven, David Herron, Steve Hone, Jan Huffman, Peter Katsoulas, Richard Kauffold, Heather McPhee, Scott Moody, John Mulcahy, Phyllis Nissen, Jacob Okyne, Donna O'Donnel, Mark Pinis, Tom Riesmeyer, Janet Russac, Cres Smith, John Smith, Judy Sommers, Bill Walsh, Richard Ward, and John Zimmerman. Thanks also to Ajit Maira and Dick Spann for their service on SPR's board of directors.

Many other colleagues work with us at SPR on special projects or as consultants. Special thanks to Allan Albrecht, the inventor of function points, for his invaluable contribution to the industry and for his outstanding work with SPR. Without Allan's pioneering work in function points, the ability to create accurate baselines and benchmarks would probably not exist.

Many thanks to Hisashi Tomino and his colleagues at Kozo Keikaku Engineering in Japan. Kozo has translated several of my prior books into Japanese. In addition, Kozo has been instrumental in the introduction of function point metrics into Japan by translating some of the relevant function point documents.

Much appreciation is due to the client organizations whose interest in software assessments, benchmarks and baselines, measurement, and process improvements have let us work together. These are the organizations whose data make estimation tools possible.

There are too many groups to name them all, but many thanks to our colleagues and clients at Andersen Consulting, AT&T, Bachman, Bellcore, Bell Northern Research, Bell Sygma, Bendix, British Air, CBIS, Charles Schwab, Church of the Latter Day Saints, Cincinnati Bell, CODEX, Computer Aid, Credit Suisse, DEC, Dunn & Bradstreet, Du Pont, EDS, Finsiel, Ford Motors, Fortis Group, General Electric, General Motors, GTE, Hartford Insurance, Hewlett-Packard, IBM, Informix, Inland Steel, Internal Revenue Service, ISSC, JC Penney,

JP Morgan, Kozo Keikaku, Language Technology, Litton, Lotus, Mead Data Central, McKinsey Consulting, Microsoft, Motorola, Nippon Telegraph, NCR, Northern Telecom, Nynex, Pacific Bell, Ralston Purina, Sapiens, Sears Roebuck, Siemens-Nixdorf, Software Publishing Corporation, SOGEL, Sun Life, Tandem, TRW, UNISYS, U.S. Air Force, U.S. Navy Surface Weapons groups, US West, Wang, Westinghouse, and many others.

Thanks also to my colleagues in software estimation: Allan Albrecht, Barry Boehm, Carol Brennan, Tom DeMarco, Don Galorath, Linda Laird, Steve McConnell, Larry Putnam, Don Reifer, Howard Rubin, Frank Freiman, Charles Symons, and Randall Jensen. While these researchers into software cost estimating may be competitors, they are also software cost-estimating pioneers and leading experts. Without their research, there would be no software cost-estimation industry. The software industry is fortunate to have researchers and authors such as these.

Appreciation is also due to those who teach software cost estimating and introduce this important topic to their students. Dr. Victor Basili and Professor Daniel Ferens have both done a great deal to bridge the gap between academia and the business world by introducing real-world estimation tools into the academic domain.

– *Capers Jones*

This page intentionally left blank

Introduction to Software Cost Estimation

Software cost estimation is a complex activity that requires knowledge of a number of key attributes about the project for which the estimate is being constructed. Cost estimating is sometimes termed “parametric estimating” because accuracy demands understanding the relationships among scores of discrete parameters that can affect the outcomes of software projects, both individually and in concert. Creating accurate software cost estimates requires knowledge of the following parameters:

- *The sizes of major deliverables, such as specifications, source code, and manuals*
- *The rate at which requirements are likely to change during development*
- *The probable number of bugs or defects that are likely to be encountered*
- *The capabilities of the development team*
- *The salaries and overhead costs associated with the development team*
- *The formal methodologies that are going to be utilized (such as the Agile methods)*
- *The tools that are going to be utilized on the project*

- *The set of development activities that are going to be carried out*
- *The cost and schedule constraints set by clients of the project being estimated*

Although the factors that influence the outcomes of software projects are numerous and some are complex, modern commercial software cost-estimation tools can ease the burden of project managers by providing default values for all of the key parameters, using industry values derived from the integral knowledge base supplied with the estimation tools.

In addition, software cost-estimation tools allow the construction of customized estimating templates that are derived from actual projects and that can be utilized for estimating projects of similar sizes and kinds.

This section discusses the origins and evolution of software cost-estimation tools and how software cost estimation fits within the broader category of software project management. In addition, this section discusses the impact of software cost-estimation tools on the success rates of software projects and uses this data to illustrate the approximate return on investment (ROI) from software cost-estimating and project management tools.

Introduction

Software cost estimating has been an important but difficult task since the beginning of the computer era in the 1940s. As software applications have grown in size and importance, the need for accuracy in software cost estimating has grown, too.

In the early days of software, computer programs were typically less than 1000 machine instructions in size (or less than 30 function points), required only one programmer to write, and seldom took more than a month to complete. The entire development costs were often less than \$5000. Although cost estimating was difficult, the economic consequences of cost-estimating errors were not very serious.

Today some large software systems exceed 25 million source code statements (or more than 300,000 function points), may require technical staffs of 1000 personnel or more, and may take more than five calendar years to complete. The development costs for such large software systems can exceed \$500 million; therefore, errors in cost estimation can be very serious indeed.

Even more serious, a significant percentage of large software systems run late, exceed their budgets, or are canceled outright due to severe underestimating during the requirements phase. In fact, excessive optimism in software cost estimation is a major source of overruns, failures, and litigation.

Software is now the driving force of modern business, government, and military operations. This means that a typical Fortune 500 corporation or a state government may produce hundreds of new applications and modify hundreds of existing applications every year. As a result of the host of software projects in the modern world, software cost estimating is now a mainstream activity for every company that builds software.

In addition to the need for accurate software cost estimates for day-to-day business operations, software cost estimates are becoming a

significant aspect in litigation. Over the past fifteen years, the author and his colleagues have observed dozens of lawsuits where software cost estimates were produced by the plaintiffs, the defendants, or both. For example, software cost estimation now plays a key part in lawsuits involving the following disputes:

- Breach of contract suits between clients and contractors
- Suits involving the taxable value of software assets
- Suits involving recovering excess costs for defense software due to scope expansion
- Suits involving favoritism in issuance of software contracts
- Suits involving wrongful termination of software personnel

From many viewpoints, software cost estimating has become a critical technology of the 21st century because software is now so pervasive.

How Software Cost-Estimating Tools Work

There are many kinds of automated tools that experienced project managers can use to create cost, schedule, and resource estimates for software projects. For example, an experienced software project manager can create a cost-estimate and schedule plan using any of the following:

- Spreadsheets
- Project management tools
- Software cost-estimating tools

A frequently asked question for software cost-estimating tool vendors is “Why do we need your tool when we already have spreadsheets and project management tools?”

The commercial software-estimating tools are differentiated from all other kinds of software project management tools and general-purpose tools, such as spreadsheets, in these key attributes:

- They contain knowledge bases of hundreds or thousands of software projects.
- They can perform size predictions, which general-purpose tools cannot.
- They can automatically adjust estimates based on tools, languages, and processes.
- They can predict quality and reliability, which general-purpose tools cannot.

- They can predict maintenance and support costs after deployment.
- They can predict (and prevent) problems long before the problems actually occur.

Unlike other kinds of project management tools, the commercial software cost-estimating tools do not depend upon the expertise of the user or project manager, although experienced managers can refine the estimates produced. The commercial cost-estimating tools contain the accumulated experience of many hundreds or thousands of software projects.

Because of the attached knowledge bases associated with commercial cost-estimating tools, novice managers or managers faced with unfamiliar kinds of projects can describe the kind of project being dealt with, and the estimating tool will construct an estimate based on similar projects derived from information contained in its associated knowledge base.

Figure 1.1 illustrates the basic principles of modern commercial software cost-estimating tools.

The starting point of software estimation is the size of the project in terms of either logical source code statements, physical lines of code, function points, or, sometimes, all three metrics. The project's size can be derived from the estimating tool's own sizing logic, supplied by users as an explicit input, or derived from analogy with similar projects stored in the estimating tool's knowledge base. Even for Agile projects and those using iterative development, at least approximate size information can be created.

Once the basic size of the project has been determined, the estimate can be produced based on the specific attributes of the project in question. Examples of the attributes that can affect the outcome of the estimate include the following:

- The rate at which project requirements may change
- The experience of the development team with this kind of project
- The process or methods used to develop the project ranging from Agile to Waterfall

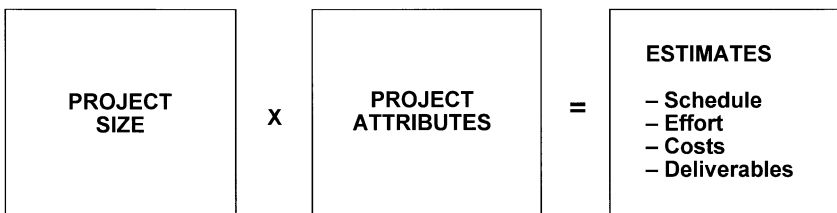


Figure 1.1 Software-estimating principles.

- The specific activities that will be performed during development
- The number of increments, iterations, or “sprints” that will be used
- The programming language or languages utilized
- The presence or absence of reusable artifacts
- The development tool suites used to develop the project
- The environment or ergonomics of the office work space
- The geographic separation of the team across multiple locations
- The schedule pressure put on the team by clients or executives
- Contractual obligations in terms of costs, dates, defects, or features

Using commercial estimating tools, these project attributes can either be supplied by the user or inherited from similar projects already stored in the estimating tool’s knowledge base. In a sense estimating tools share some of the characteristics of the object-oriented paradigm in that they allow inheritance of shared attributes from project to project.

In software-estimating terminology, these shared attributes are termed *templates*, and they can be built in a number of ways. For example, estimating-tool users can point to an existing completed project and select any or all of the features of that project as the basis of the template. Thus, if the project selected as the basis of a template were a systems software project, used the C programming language, and utilized formal design and code inspections, these attributes could be inherited as part of the development cycle and could become part of an estimating template for other projects.

Many other attributes from historical projects can also be inherited and can become aspects of software-estimating templates. For example, a full estimating template can contain inherited attribute data on such topics as the following:

- The experience of the development team in similar applications
- The process or methodology used to develop the application
- The SEI capability maturity level of the organization
- The standards that will be adhered to, such as ISO, DoD, IEEE, and so forth
- The tools used during design, coding, testing, and so forth
- The programming language or languages utilized
- The volumes of reusable artifacts available
- The ergonomics of the programming office environment

Since software projects are not identical, any of these inherited attributes can be modified as the need arises. However, the availability of

templates makes the estimation process quicker, more convenient, and more reliable because templates substitute specific knowledge from local projects for generic industry default values.

Templates can also be derived from sets of projects rather than from one specific project, or can even be custom-built by the users, using artificial factors. However, the most common method of template development is to use the automatic template construction ability of the estimating tool, and simply select relevant historical projects to be used as the basis for the template.

As a general rule, software-estimating templates are concerned with four key kinds of inherited attributes: (1) *personnel*, (2) *technologies*, (3) *tools*, and (4) the *programming environment*, as illustrated by Figure 1.2.

Three of these four factors—the experience of the personnel, the development process, and the technology (programming languages and support tools)—are fairly obvious in their impact. What is not obvious, but is equally important, is the impact of the fourth factor—*environment*.

The environment factor covers individual office space and the communication channels among geographically dispersed development teams. Surprisingly, access to a quiet, noise-free office environment is one of the major factors that influences programming productivity.

The ability to include ergonomic factors in an estimate is an excellent example of the value of commercial software cost-estimating tools. Not only do they contain the results of hundreds or thousands of completed projects, but the tools contain data about influential factors that many human project managers may not fully understand.

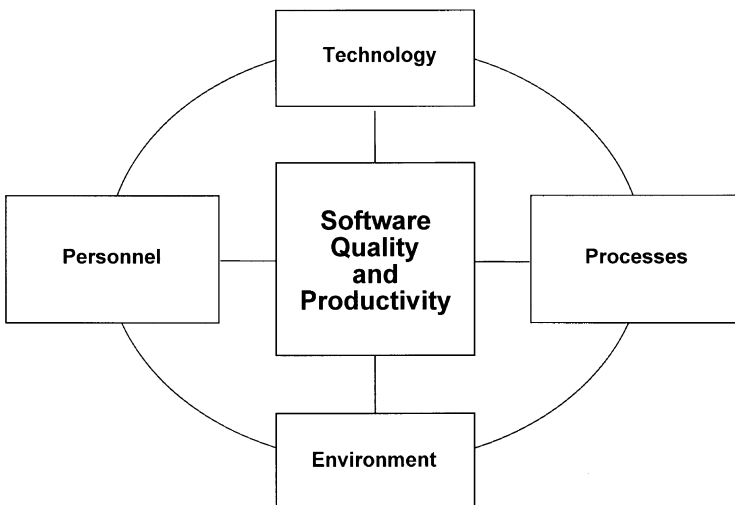


Figure 1.2 Key estimate factors.

The four key sets of attributes must be considered whether estimating manually or using an automated estimating tool. However, one of the key features of commercial software-estimating tools is the fact that they are repositories containing the results of hundreds or thousands of software projects, and so the effect of these four attribute areas can be examined, and their impacts can be analyzed.

There is a standard sequence for software cost estimation, which the author has used for more than 35 years. This sequence can be used with manual software cost estimates, and also mirrors the estimation stages in the software-estimation tools that the author has designed. There are ten steps in this sequence, although the sequence starts with 0 because the first stage is a pre-estimate analysis of the requirements of the application.

Step 0: Analyze the Requirements

Software cost estimation at the project level cannot be performed unless the requirements of the project are well understood. Therefore, before estimating itself can begin, it is necessary to explore and understand the user requirements. At some point in the future it should be possible to create estimates automatically from the requirements specifications, but the current level of estimating technology demands human intervention.

A common estimating activity today is to analyze the software requirements and create function point totals based on those requirements. This provides the basic size data used for formal cost estimation. Function point analysis is usually performed manually by certified function point counting personnel. A time is rapidly approaching when function point totals can be derived automatically from software requirements, and this method may appear in commercial software cost-estimation tools within a few years.

It is a known fact that requirements for large systems cannot be fully defined at the start of the project. This fact is the basis for the Agile methods, extreme programming, Scrum, and a number of others. This fact is also embedded in the algorithms for several commercial software estimating tools. Once the initial requirements are understood, the average rate of growth of new requirements is about 2 percent per calendar month. This growth can be planned for and included in the estimate.

Step 1: Start with Sizing

Every form of estimation and every commercial software cost-estimating tool needs the sizes of key deliverables in order to complete an estimate. Size data can be derived in several fashions, including the following:

- Size prediction using an estimating tool's built-in sizing algorithms
- Sizing by extrapolation from function point totals
- Sizing by analogy with similar projects of known size
- Guessing at the size using "project manager's intuition"
- Guessing at the size using "programmer's intuition"
- Sizing using statistical methods or Monte Carlo simulation

For Agile methods and those projects using iterative development, sizing of the entire application may be deferred until the early increments are complete. However, even for Agile and iterative projects it is possible to make an approximate prediction of final size just by comparing the nature of the project to similar projects or using size approximations based on the class, type, and nature of the software. Later in this book examples are given of such sizing methods.

The basic size of the application being estimated is usually expressed in terms of function points, source code statements, or both. However, it is very important to size all deliverables and not deal only with code. For example, more than 50 kinds of paper documents are associated with large software projects, and they need to be sized also. Of course, when using one of the commercial software estimating tools that support documents sizing, these sizes will automatically be predicted.

Source code sizing must be tailored to specific programming languages, and more than 600 languages are now in use. About one-third of software projects utilize more than a single programming language. More than a dozen kinds of testing occur, and each will require different volumes of test cases.

Sizing is a key estimating activity. If the sizes of major deliverables can be predicted within 5 to 10 percent, then the accuracy of the overall estimate can be quite good. If size predictions are wildly inaccurate, then the rest of the estimate will be inaccurate, too. As mentioned earlier, empirical evidence from large software projects indicates that requirements grow at an average rate of about 2 percent per calendar month from the end of the requirements phase until the start of the testing phase. The total growth of requirements can exceed 50 percent of the volume of the initial requirements when measured with function points. A major problem with estimating accuracy has been ignoring or leaving out requirements creep. Modern cost-estimating tools can predict requirements growth and can include their costs and schedules in the estimate.

The technologies available for sizing have been improving rapidly. In the early days of software cost estimation, size data had to be supplied by users, using very primitive methods. Now modern software

cost-estimating tools have a number of sizing capabilities available, including support for very early size estimates even before the requirements are firm.

Step 2: Identify the Activities to Be Included

Once the sizes of key deliverables are known, the next step is to select the set of activities that are going to be performed. In this context the term *activities* refers to the work that will be performed for the project being estimated, such as requirements, internal design, external design, design inspections, coding, code inspections, user document creation, meetings or Scrum sessions, change control, integration, quality assurance, unit testing, new function testing, regression testing, system testing, and project management. Accurate estimation is impossible without knowledge of the activities that are going to be utilized.

Activity patterns vary widely from project to project. Large systems utilize many more activities than do small projects. Waterfall projects utilize more activities than Agile projects. For projects of the same size, military and systems software utilize more activities than do information systems. Local patterns of activities are the ones to utilize, because they reflect your own enterprise's software development methodologies.

Modern software cost-estimating tools have built-in logic for selecting the activity patterns associated with many kinds of software development projects. Users can also adjust the activity patterns to match local variations.

Step 3: Estimate Software Defect Potentials and Removal Methods

The most expensive and time-consuming work of software development is the work of finding bugs and fixing them. In the United States the number of bugs or defects in requirements, design, code, documents, and bad-fixes averages five per function point. Average defect removal efficiency before delivery is 85 percent. The cost of finding and repairing these defects averages about 35 percent of the total development cost of building the application. The schedule time required is about 30 percent of project development schedules. Defect repair costs and schedules are often larger than coding costs and schedules. Accuracy in software cost estimates is not possible if defects and defect removal are not included in the estimates. The author's software cost-estimating tools include full defect-estimation capabilities, and support all known kinds of defect-removal activity. This is necessary because the total effort, time, and cost devoted to a full series of reviews, inspections, and multistage tests will cost far more than source code itself.

Defect estimation includes predictive abilities for requirements defects, design defects, coding defects, user documentation defects, and a very troubling category called *bad fix defects*. The phrase *bad fix* refers to new defects accidentally injected as a by-product of repairing previous defects. Bad fix defects average about 7 percent of all defect repairs. Some estimating tools can predict bad fixes.

Estimating tools that support commercial software can also predict duplicate defect reports, or bugs found by more than one customer. It is also possible to estimate *invalid defect reports*, or bug reports that turn out not to be the fault of the software, such as user errors or hardware problems.

The ability to predict software defects would not be very useful without another kind of estimation, which is predicting the *defect-removal efficiency* of various kinds of reviews, inspections, and test stages. Modern software cost-estimating tools can predict how many bugs will be found by every form of defect removal, from desk checking through external beta testing.

Step 4: Estimate Staffing Requirements

Every software deliverable has a characteristic *assignment scope*, or amount of work that can be done by a single employee. For example, an average assignment for an individual programmer will range from 5000 to 15,000 source code statements (from about 50 up to 2000 function points).

However, large systems also utilize many specialists, such as system architects, database administrators, quality assurance specialists, software engineers, technical writers, testers, and the like. Identifying each category of worker and the numbers of workers for the overall project is the next step in software cost estimation.

Staffing requirements depend upon the activities that will be performed and the deliverables that will be created, so staffing predictions are derived from knowledge of the overall size of the application and the activity sets that will be included. Staffing predictions also need to be aware of “pair programming” or two-person teams, which are part of some of the new Agile methods.

For large systems, programmers themselves may comprise less than half of the workforce. Various kinds of specialists and project managers comprise the other half. Some of these specialists include quality assurance personnel, testing personnel, technical writers, systems analysts, database administrators, and configuration control specialists. If the project is big enough to need specialists, accurate estimation requires that their efforts be included. Both programming and other kinds of noncoding activities,

such as production of manuals and quality assurance, must be included to complete the estimate successfully.

Step 5: Adjust Assumptions Based on Capabilities and Experience

Software personnel can range from top experts with years of experience to rank novices on their first assignment. Once the categories of technical workers have been identified, the next step is to make adjustments based on typical experience levels and skill factors.

Experts can take on more work, and perform it faster, than can novices. This means that experts will have larger assignment scopes and higher production rates than average or inexperienced personnel.

Other adjustments include work hours per day, vacations and holidays, unpaid and paid overtime, and assumptions about the geographic dispersal of the software team. Adjusting the estimate to match the capabilities of the team is one of the more critical estimating activities.

While estimating tools can make adjustments to match varying degrees of expertise, these tools have no way of knowing the specific capabilities of any given team. Many commercial estimating tools default to “average” capabilities, and allow users to adjust this assumption upward or downward to match specific team characteristics.

Step 6: Estimate Effort and Schedules

Effort and schedule estimates are closely coupled, and often are performed in an iterative manner.

Accurate effort estimation requires knowledge of the basic size of the application plus the numbers and experience levels of the software team members and the sizes of various deliverables they are expected to produce, such as specifications and user manuals.

Accurate schedule estimation requires knowledge of the activities that will be performed, the number of increments or “sprints” that will be carried out, the sizes of various deliverables, the overlap between activities with mutual dependencies, and the numbers and experience levels of the software team members.

Schedule and effort estimates are closely coupled, but the interaction between these two dimensions is complicated and sometimes is counterintuitive. For example, if a software project will take six months if it is developed by one programmer, adding a second programmer will not cut the schedule to three months. Indeed, a point can be reached where putting on additional personnel may slow down the project’s schedule rather than accelerating it.

The complex sets of rules that link effort and schedules for software projects are the heart of the algorithms for software cost-estimating tools.

As an example of one of the more subtle rules that estimating tools contain, adding personnel to a software project within one department will usually shorten development schedules. But if enough personnel are added so that a second department is involved, schedules will stretch out. The reason for this is that software schedules, and also productivity rates, are inversely related to the number of project managers engaged. There are scores of rules associated with the interaction of schedules and effort, and some of these are both subtle and counterintuitive.

In fact, for very large software projects with multiple teams, the rate at which development productivity declines tends to correlate more closely to the number of managers that are engaged than to the actual number of programmers involved. This phenomenon leads to some subtle findings, such as the fact that projects with a small span of control (less than six developers per manager) may have lower productivity than similar projects with a large span of control (12 developers per manager).

Step 7: Estimate Development Costs

Development costs are the next-to-last stage of estimation and are very complex. Development costs are obviously dependent upon the effort and schedule for software projects, so these factors are predicted first, and then costs are applied afterwards.

Costs for software projects that take exactly the same amount of effort in terms of hours or months can vary widely due to the following causes:

- Average salaries of workers and managers on the project
- The corporate burden rate or overhead rate applied to the project
- Inflation rates, if the project will run for several years
- Currency exchange rates, if the project is developed internationally

There may also be special cost topics that will have to be dealt with separately, outside of the basic estimate:

- License fees for any acquired software needed
- Capital costs for any new equipment
- Moving and living costs for new staff members
- Travel costs for international projects or projects developed in different locations
- Contractor and subcontractor costs
- Legal fees for copyrights, patents, or other matters
- Marketing and advertising costs

- Costs for developing videos or CD-ROM tutorial materials and training
- Content acquisition costs if the application is a web-based product

On the whole, developing a full and complete cost estimate for a software project is much more complex than simply developing a resource estimate of the number of work hours that are likely to be needed. Many cost elements, such as burden rates or travel, are only indirectly related to effort and can impact the final cost of the project significantly.

The normal pattern of software estimation is to use hours, days, weeks, or months of effort as the primary estimating unit, and then apply costs at the end of the estimating cycle once the effort patterns have been determined.

Step 8: Estimate Maintenance and Enhancement Costs

Software projects often continue to be used and modified for many years. Maintenance and enhancement cost estimation are special topics, and are more complex than new project cost estimation.

Estimating maintenance costs requires knowledge of the probable number of users of the application, combined with knowledge of the probable number of bugs or defects in the product at the time of release.

Estimating enhancement costs requires good historical data on the rate of change of similar projects once they enter production and start being used. For example, new software projects can add 10 percent or more in total volume of new features with each release for several releases in a row, but then slow down for a period of two to three years before another major release occurs.

Many commercial estimating tools can estimate both the initial construction costs of a project and the maintenance and enhancement cost patterns for more than five years of usage by customers.

There is no actual limit on the number of years that can be estimated, but because long-range projections of user numbers and possible new features are highly questionable, the useful life of maintenance and enhancement estimates runs from three to five years. Estimating maintenance costs ten years into the future can be done, but no estimate of that range can be regarded as reliable because far too many uncontrollable business variables can occur.

Step 9: Present Your Estimate to the Client and Defend It Against Rejection

Once a cost estimate is prepared, the next step is to present the estimate to the client who is going to fund the project. For large systems and

applications of 5000 function points or larger (equivalent to roughly 500,000 source code statements) about 60 percent of the initial estimates will be rejected by the client. Either the estimated schedule will be too long, the costs will be too high, or both. Often the client will decree a specific delivery date much shorter than the estimated date. Often the client will decree that costs must be held within limits much lower than the estimated costs. Projects where formal estimates are rejected and replaced by arbitrary schedules and costs derived from external business needs rather than team capabilities have the highest failure rates in the industry. About 60 percent of such projects will be cancelled and never completed at all. (At the point of cancellation, both costs and schedules will already have exceeded their targets.) Of the 40 percent of projects that finally do get completed, the average schedule will be about one year late, and the average cost will be about 50 percent higher than targets.

The best defense against having a cost estimate rejected is to have solid historical data from at least a dozen similar projects. The second-best defense against have a cost estimate rejected is to prepare a full activity-based estimate that includes quality, paperwork, requirements creep, all development activities, and all maintenance tasks. You will need to prove that your estimate has been carefully prepared and has left nothing to chance. High-level or phase-level estimates that lack detail are not convincing and are easy to reject.

Cautions About Accidental Omissions from Estimates

Because software-estimating tools have such an extensive knowledge base, they are not likely to make the kinds of mistakes that inexperienced human managers make when they create estimates by hand or with general-purpose tools and accidentally omit activities from the estimate.

For example, when estimating large systems, coding may be only the fourth most expensive activity. Human managers often tend to leave out or underestimate the non-code work, but estimating tools can include these other activities.

- Historically, the effort devoted to finding and fixing bugs by means of reviews, walkthroughs, inspections, and testing takes more time and costs more than any other software activities. Therefore, accurate cost estimates need to start with quality predictions, because defect-removal costs are often more expensive than anything else.
- In second place as major cost elements are the expenses and effort devoted to the production of paper documents, such as plans, specifications, user manuals, and the like. For military software projects,

paperwork will cost twice as much as the code itself. For large civilian projects greater than 1000 function points or 100,000 source code statements, paper documents will be a major cost element and will approach or exceed the cost of the code.

- In third place for many large projects are the costs and schedules of dealing with “creeping requirements” or new features added to the project after the requirements phase. All software projects will grow due to creeping requirements and therefore this factor should be an integral part of the estimates for all major software projects.
- For some large distributed applications that involve multiple development locations or subcontractors, the costs of meetings and travel among the locations can cost more than the source code and may be in fourth place in the sequence of all software costs. A frequent omission from software cost estimates is the accidental exclusion of travel costs (airlines, hotels, etc.) for meetings among the development teams that are located in different cities or different countries. For very large kinds of systems, such as operating systems, telecommunication systems, or defense systems, which may involve distributed development in half a dozen countries and a dozen cities, the costs of travel can exceed the cost of coding significantly, and this topic should not be left out by accident.
- Many software cost estimates—and many measurement systems, too—cover only the core activities of software development and ignore such topics as project management and support (i.e., program librarians, secretaries, administration, etc.). These ancillary activities are part of the project and can, in some cases, top 20 percent of total costs. This is far too much to leave out by accident.
- The software domain has fragmented into a number of specialized skills and occupations. It is very common to accidentally leave out the contributions of specialists if their skills are needed only during portions of a software development cycle. Some of the specialist groups that tend to be accidentally omitted from software cost estimates include quality assurance specialists, technical writing specialists, function point specialists, database administration specialists, performance tuning specialists, network specialists, and system administration specialists. The combined contributions of these and other specialists may total more than 20 percent of all software development costs and should not be omitted by accident.
- The most common omission from internal software cost estimates for information systems are the costs expended by users during requirements definition, prototyping, status reviews, phase reviews, documentation, inspections, acceptance testing, and other activities where

the developers have a key role. Since user representatives are not usually considered to be part of the project team, their contributions to the project are seldom included in software cost estimates, and are seldom included in measurement studies, either. The actual amount of effort contributed by users to major software development projects can approach 20 percent of the total work in some cases, which is not a trivial amount and is far too significant to leave out by accident. Some commercial software cost-estimating tools keep a separate chart of accounts for user activities and allow user efforts to be added to total project costs, if desired.

- For many projects, maintenance after delivery quickly costs more than the development of the application itself. It is unwise to stop the estimate at the point of delivery of the software without including at least five years of maintenance and enhancement estimates. Since maintenance (defect repairs) and enhancements (adding new features) have different funding sources, many estimating tools separate these two activities. Other forms of maintenance work, such as customer support or field service, may also be included in post-release estimates.

A key factor that differentiates modern commercial software cost-estimating tools from general-purpose tools, such as spreadsheets and project management tools, is the presence of full life-cycle historical data. This gives them the ability to estimate quality and to estimate the sizes and costs of producing paper deliverables, the probable volumes of creeping requirements, and the costs of coding and testing.

When considering acquisition of a software cost-estimating tool, be sure that the knowledge base includes the kind of software you are interested in. The real-life cost and schedule results of information systems, systems software, commercial software, military software, and embedded software are not identical, and you need to be sure the estimating tool contains data on the kinds of software you are concerned with. Some tools support all classes of software, but others are more narrow in focus.

Software Cost Estimating and Other Development Activities

Software cost estimating is not a “standalone” activity. The estimates are derived in large part from the requirements of the project, and will be strongly affected by the tools, process, and other attributes associated with the project. A cost estimate is a precursor for departmental budgets, and also serves as a baseline document for comparing accumulated costs against projected costs.

For any project larger than trivial, multiple cost estimates will be prepared during the course of development, including but not limited to the following:

- A rough pre-requirements guesstimate
- An initial formal estimate derived from the project requirements
- One or more midlife estimates, which reflect requirements changes
- A final cost accumulation using project historical data

In addition, since the software industry is somewhat litigious, cost estimates may also be prepared as a by-product of several kinds of litigation, including the following:

- Litigation for breach of contract between software clients and out-source companies
- Litigation involving the taxable value of software in tax disputes

In the course of developing a software project, historical data will steadily be accumulated. This means that after the first rough guesstimate and the initial requirements estimate, future estimates will need to interleave historical data with predicted data. Therefore, software-estimating tools need the ability to capture historical data and to selectively display both historical data and predicted values.

Figure 1.3 illustrates how software cost estimation fits into the context of other key software development activities.

As can be seen from Figure 1.3, estimating is closely aligned with other key development phases. When done well, software cost estimates are among the most valuable documents in the entire software world, because they make a software project real and tangible in terms of the resources, schedules, and costs that will be required.

However, cost estimates that are poorly constructed and grossly inaccurate are key factors in almost every major software disaster. The best advice for those charged with constructing software cost estimates is the following:

- Be accurate.
- Be conservative.
- Base the estimate on solid historical data.
- Include quality, since software quality affects schedules and costs.
- Include paper documents, since they can cost more than source code.
- Include project management.

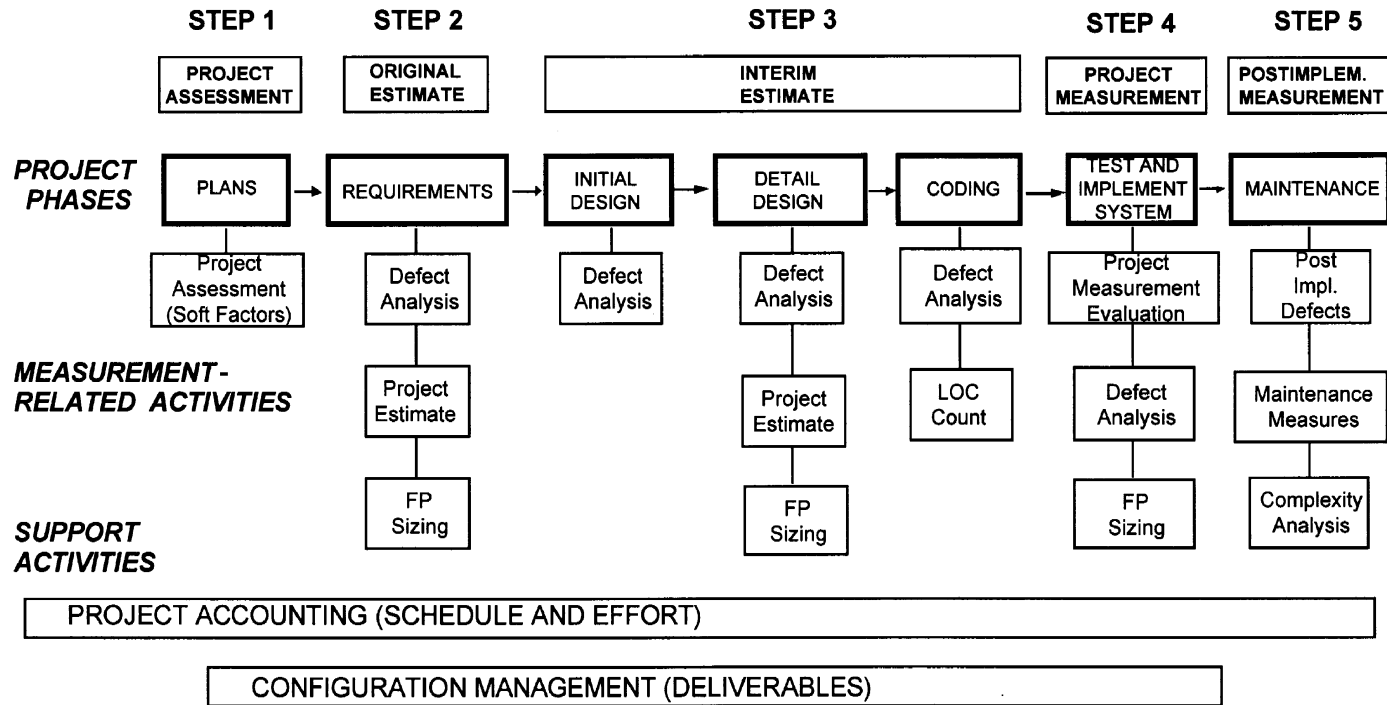


Figure 1.3 Software cost estimation and other activities.

- Include the effects of creeping requirements.
- Do not exaggerate the effect of tools, languages, or methods.
- Get below phases to activity-level cost estimates.
- Be prepared to defend the assumptions of your estimate.

Even with the best estimating tools, accurate software cost estimating is complicated and can be difficult. But without access to good historical data, accurate software cost estimating is almost impossible. Measurement and estimation are twin technologies and both are urgently needed by software project managers.

Measurement and estimation are also linked in the commercial software cost-estimation marketplace, since many of the commercial estimating companies are also benchmark and measurement companies. As better historical data becomes available, the features of the commercial software cost-estimating tools are growing stronger.

References

- Barrow, Dean, Susan Nilson, and Dawn Timberlake: *Software Estimation Technology Report*, Air Force Software Technology Support Center, Hill Air Force Base, Utah, 1993.
- Boehm, Barry: *Software Engineering Economics*, Prentice-Hall, Englewood Cliffs, N.J., 1981.
- : *Software Cost Estimation with COCOMO II*, Prentice-Hall, Englewood Cliffs, NJ;
- Brown, Norm (ed.): *The Program Manager's Guide to Software Acquisition Best Practices*, Version 1.0, U.S. Department of Defense, Washington, D.C., July 1995.
- Charette, Robert N.: *Software Engineering Risk Analysis and Management*, McGraw-Hill, New York, 1989.
- : *Application Strategies for Risk Analysis*, McGraw-Hill, New York, 1990.
- Cohn, Mike: *Agile Estimating and Planning* (Robert C. Martin Series), Prentice-Hall PTR, Englewood Cliffs, NJ, 2005.
- Coombs, Paul: *IT Project Estimation: A Practical Guide to the Costing of Software*, Cambridge University Press, Melbourne, Australia.
- DeMarco, Tom: *Controlling Software Projects*, Yourdon Press, New York.
- : *Deadline*, Dorset House Press, New York, 1997.
- Department of the Air Force: *Guidelines for Successful Acquisition and Management of Software Intensive Systems*; vols. 1 and 2, Software Technology Support Center, Hill Air Force Base, Utah, 1994.
- Dreger, Brian: *Function Point Analysis*, Prentice-Hall, Englewood Cliffs, N.J., 1989.
- Galorath, Daniel D. and Michael W. Evans: *Software Sizing, Estimation, and Risk Management*, Auerbach, Philadelphia, PA, 2006.
- Garmus, David, and David Herron: *Measuring the Software Process: A Practical Guide to Functional Measurement*, Prentice-Hall, Englewood Cliffs, N.J., 1995.
- Garmus, David and David Herron: *Function Point Analysis: Measurement Practices for Successful Software Projects*, Addison-Wesley, Boston, Mass., 2001.
- Grady, Robert B.: *Practical Software Metrics for Project Management and Process Improvement*, Prentice-Hall, Englewood Cliffs, N.J., 1992.
- and Deborah L. Caswell: *Software Metrics: Establishing a Company-Wide Program*, Prentice-Hall, Englewood Cliffs, N.J., 1987.

- Gullede, Thomas R., William P. Hutzler, and Joan S. Lovelace (eds.): *Cost Estimating and Analysis—Balancing Technology with Declining Budgets*, Springer-Verlag, New York, 1992.
- Howard, Alan (ed.): *Software Metrics and Project Management Tools*, Applied Computer Research (ACR), Phoenix, Ariz., 1997.
- IFPUG Counting Practices Manual, Release 4, International Function Point Users Group, Westerville, Ohio, April 1995.
- Jones, Capers: *Critical Problems in Software Measurement*, Information Systems Management Group, 1993a.
- : *Software Productivity and Quality Today—The Worldwide Perspective*, Information Systems Management Group, 1993b.
- : *Assessment and Control of Software Risks*, Prentice-Hall, Englewood Cliffs, N.J., 1994.
- : *New Directions in Software Management*, Information Systems Management Group.
- : *Patterns of Software System Failure and Success*, International Thomson Computer Press, Boston, 1995.
- : *Applied Software Measurement*, 2nd ed., McGraw-Hill, New York, 1996.
- : *The Economics of Object-Oriented Software*, Software Productivity Research, Burlington, Mass., April 1997a.
- : *Software Quality—Analysis and Guidelines for Success*, International Thomson Computer Press, Boston, 1997b.
- : *The Year 2000 Software Problem—Quantifying the Costs and Assessing the Consequences*, Addison-Wesley, Reading, Mass., 1998.
- : *Software Assessments, Benchmarks, and Best Practices*, Addison-Wesley, Boston, Mass., 2000.
- Kan, Stephen H.: *Metrics and Models in Software Quality Engineering*, 2nd edition, Addison-Wesley, Boston, Mass., 2003.
- Kemerer, C. F.: “Reliability of Function Point Measurement—A Field Experiment,” *Communications of the ACM*, **36**: 85–97 (1993).
- Keys, Jessica: *Software Engineering Productivity Handbook*, McGraw-Hill, New York, 1993.
- Laird, Linda M. and Carol M. Brennan: *Software Measurement and Estimation: A practical Approach*, John Wiley & Sons, New York, 2006.
- Lewis, James P.: *Project Planning, Scheduling & Control*, McGraw-Hill, New York, New York, 2005.
- Marciniak, John J. (ed.): *Encyclopedia of Software Engineering*, vols. 1 and 2, John Wiley & Sons, New York, 1994.
- McConnell, Steve: *Software Estimation: Demystifying the Black Art*, Microsoft Press, Redmond, WA, 2006.
- Mertes, Karen R.: *Calibration of the CHECKPOINT Model to the Space and Missile Systems Center (SMC) Software Database (SWDB)*, Thesis AFIT/GCA/LAS/96S-11, Air Force Institute of Technology (AFIT), Wright-Patterson AFB, Ohio, September 1996.
- Oourada, Gerald, and Daniel V. Ferens: “Software Cost Estimating Models: A Calibration, Validation, and Comparison,” in *Cost Estimating and Analysis: Balancing Technology and Declining Budgets*, Springer-Verlag, New York, 1992, pp. 83–101.
- Perry, William E.: *Handbook of Diagnosing and Solving Computer Problems*, TAB Books, Blue Ridge Summit, Pa., 1989.
- Pressman, Roger: *Software Engineering: A Practitioner’s Approach with Bonus Chapter on Agile Development*, McGraw-Hill, New York, 2003.
- Putnam, Lawrence H.: *Measures for Excellence—Reliable Software on Time, Within Budget*, Yourdon Press/Prentice-Hall, Englewood Cliffs, N.J., 1992.
- , and Ware Myers: *Industrial Strength Software—Effective Management Using Measurement*, IEEE Press, Los Alamitos, Calif., 1997.
- Reifer, Donald (ed.): *Software Management*, 4th ed., IEEE Press, Los Alamitos, Calif., 1993.

- Rethinking the Software Process*, CD-ROM, Miller Freeman, Lawrence, Kans., 1996. (This CD-ROM is a book collection jointly produced by the book publisher, Prentice-Hall, and the journal publisher, Miller Freeman. It contains the full text and illustrations of five Prentice-Hall books: *Assessment and Control of Software Risks* by Capers Jones; *Controlling Software Projects* by Tom DeMarco; *Function Point Analysis* by Brian Dreger; *Measures for Excellence* by Larry Putnam and Ware Myers; and *Object-Oriented Software Metrics* by Mark Lorenz and Jeff Kidd.)
- Rubin, Howard: *Software Benchmark Studies for 1997*, Howard Rubin Associates, Pound Ridge, N.Y., 1997.
- Roetzheim, William H., and Reyna A. Beasley: *Best Practices in Software Cost and Schedule Estimation*, Prentice-Hall PTR, Upper Saddle River, N.J., 1998.
- Stukes, Sherry, Jason Deshoretz, Henry Apgar, and Ilona Macias: *Air Force Cost Analysis Agency Software Estimating Model Analysis—Final Report*, TR-9545/008-2, Contract F04701-95-D-0003, Task 008, Management Consulting & Research, Inc., Thousand Oaks, Calif., September 1996.
- Stutzke, Richard D.: *Estimating Software-Intensive Systems: Projects, Products, and Processes*, Addison-Wesley, Boston, Mass, 2005.
- Symons, Charles R.: *Software Sizing and Estimating—Mk II FPA (Function Point Analysis)*, John Wiley & Sons, Chichester, U.K., 1991.
- Wellman, Frank: *Software Costing: An Objective Approach to Estimating and Controlling the Cost of Computer Software*, Prentice-Hall, Englewood Cliffs, N.J., 1992.
- Yourdon, Ed: *Death March—The Complete Software Developer's Guide to Surviving "Mission Impossible" Projects*, Prentice-Hall PTR, Upper Saddle River, N.J., 1997.
- Zells, Lois: *Managing Software Projects—Selecting and Using PC-Based Project Management Systems*, QED Information Sciences, Wellesley, Mass., 1990.
- Zvegintzov, Nicholas: *Software Management Technology Reference Guide*, Dorset House Press, New York, 1994.