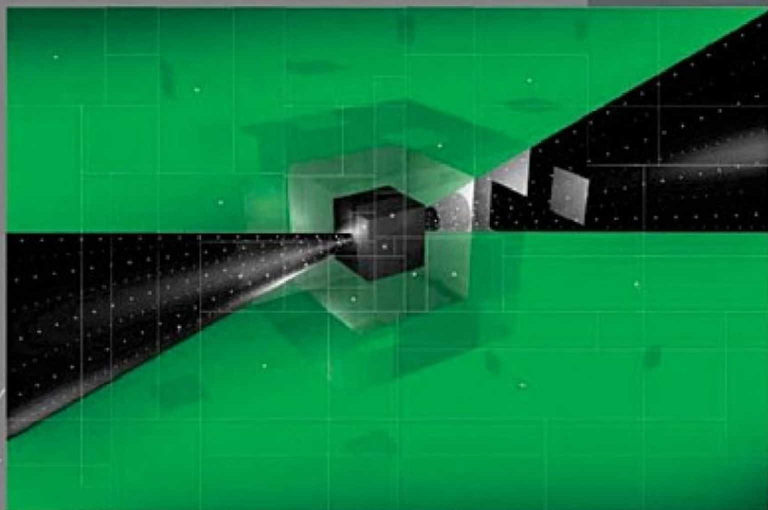
Microsoft

BEST PRACTICES

SOFTWARE ESTIMATION



Demystifying the Black Art

Steve McConnell

Two-time winner of *Software Development* magazine's Jolt Award

Table of Contents

Welcome

Art vs. Science of Software Estimation

Why This Book Was Written and Who It Is For

Key Benefits Of This Book

What This Book Is Not About

Where to Start

Acknowledgments

Equations

Figures

I. Critical Estimation Concepts

1. What Is an "Estimate"?

1.1. Estimates, Targets, and Commitments

1.2. Relationship Between Estimates and Plans

1.3. Communicating about Estimates, Targets, and Commitments

1.4. Estimates as Probability Statements

1.5. Common Definitions of a "Good" Estimate

1.6. Estimates and Project Control

1.7. Estimation's Real Purpose

Copyrighted material

1.8. A Working Definition of a “Good Estimate”

Additional Resources

2. How Good an Estimator Are You?

2.1. A Simple Estimation Quiz

2.2. Discussion of Quiz Results

How Confident Is “90% Confident”?

How Wide Should You Make Your Ranges?

Where Does Pressure to Use Narrow Ranges Come From?

How Representative Is This Quiz of Real Software Estimates?

3. Value of Accurate Estimates

3.1. Is It Better to Overestimate or Underestimate?

Arguments Against Overestimation

Arguments Against Underestimation

Weighing the Arguments

3.2. Details on the Software Industry’s Estimation Track Record

How Late Are the Late Projects?

One Company’s Experience

The Software Industry’s Systemic Problem

3.3. Benefits of Accurate Estimates

3.4. Value of Predictability Compared with Other Desirable Project Attributes

3.5. Problems with Common Estimation Techniques

Copyrighted material

Additional Resources

4. Where Does Estimation Error Come From?

4.1. Sources of Estimation Uncertainty

4.2. The Cone of Uncertainty

Can You Beat the Cone?

The Cone Doesn’t Narrow Itself

Accounting for the Cone of Uncertainty in Software Estimates

Relationship Between the Cone of Uncertainty and Commitment

The Cone of Uncertainty and Iterative Development

4.3. Chaotic Development Processes

4.4. Unstable Requirements

Estimating Requirements Growth

4.5. Omitted Activities

4.6. Unfounded Optimism

4.7. Subjectivity and Bias

4.8. Off-the-Cuff Estimates

4.9. Unwarranted Precision

4.10. Other Sources of Error

Additional Resources

5. Estimate Influences

5.1. Project Size

Why Is This Book Discussing Size in Lines of Code?

Copyrighted material

Diseconomies of Scale
When You Can Safely Ignore Diseconomies of Scale
Importance of Diseconomy of Scale in Software Estimation
5.2. Kind of Software Being Developed
5.3. Personnel Factors
5.4. Programming Language
5.5. Other Project Influences
5.6. Diseconomies of Scale Revisited
Additional Resources
II. Fundamental Estimation Techniques
6. Introduction to Estimation Techniques
6.1. Considerations in Choosing Estimation Techniques
What's Being Estimated
Project Size
Software Development Style
Effect of Development Style on Choice of Estimation Techniques
Development Stage
Accuracy Possible
6.2. Technique Applicability Tables
7. Count, Compute, Judge
7.1. Count First

Copyrighted material

7.2. What to Count

7.3. Use Computation to Convert Counts to Estimates

7.4. Use Judgment Only as a Last Resort

Additional Resources

8. Calibration and Historical Data

8.1. Improved Accuracy and Other Benefits of Historical Data

Accounts for Organizational Influences

Avoids Subjectivity and Unfounded Optimism

Reduces Estimation Politics

8.2. Data to Collect

Issues Related to Size Measures

Issues Related to Effort Measures

Issues Related to Calendar Time Measures

Issues Related to Defect Measures

Other Data Collection Issues

8.3. How to Calibrate

8.4. Using Project Data to Refine Your Estimates

8.5. Calibration with Industry Average Data

8.6. Summary

Additional Resources

9. Individual Expert Judgment

9.1. Structured Expert Judgment

Copyrighted material

Who Creates the Estimates?

Granularity

Use of Ranges

Formulas

Checklists

9.2. Compare Estimates to Actuals

Additional Resources

10. Decomposition and Recomposition

10.1. Calculating an Accurate Overall Expected Case

The Law of Large Numbers

How Small Should the Estimated Pieces Be?

10.2. Decomposition via an Activity-Based Work Breakdown Structure

10.3. Hazards of Adding Up Best Case and Worst Case Estimates

Warning: Math Ahead!

What Went Wrong?

10.4. Creating Meaningful Overall Best Case and Worst Case Estimates

Computing Aggregate Best and Worst Cases for Small Numbers of Tasks (Simple Standard Deviation Formula)

Computing Aggregate Best and Worst Cases for Large Numbers of Tasks (Complex Standard Deviation Formula)

Creating the Aggregate Best and Worst Case Estimates

Copyrighted material

Cautions About Percentage Confident Estimates

Additional Resources

11. Estimation by Analogy

11.1. Basic Approach to Estimating by Analogy

Step 1: Get Detailed Size, Effort, and Cost Results for a Similar Previous Project

Step 2: Compare the Size of the New Project to a Similar Past Project

Step 3: Build Up the Estimate for the New Project's Size as a Percentage of the Old Project's Size

Step 4: Create an Effort Estimate Based on the Size of the New Project Compared to the Previous Project

Step 5: Check for Consistent Assumptions Across the Old and New Projects

11.2. Comments on Uncertainty in the Triad Estimate

Estimation Uncertainty, Plans, and Commitments

12. Proxy-Based Estimates

12.1. Fuzzy Logic

How to Get the Average Size Numbers

How to Classify New Functionality

How Not to Use Fuzzy Logic

Extensions of Fuzzy Logic

12.2. Standard Components

Copyrighted material

Using Standard Components with Percentiles

Limitations of Standard Components

12.3. Story Points

Cautions About Ratings Scales

12.4. T-Shirt Sizing

12.5. Other Uses of Proxy-Based Techniques

12.6. Additional Resources

13. Expert Judgment in Groups

13.1. Group Reviews

13.2. Wideband Delphi

Effectiveness of Wideband Delphi

"The Truth Is Out There"

When to Use Wideband Delphi

Additional Resources

14. Software Estimation Tools

14.1. Things You Can Do with Tools That You Can't Do Manually

14.2. Data You'll Need to Calibrate the Tools

14.3. One Thing You Shouldn't Do with a Tool Any More than You Should Do Otherwise

14.4. Summary of Available Tools

Additional Resources

15. Use of Multiple Approaches

Copyrighted material

Additional Resources

16. Flow of Software Estimates on a Well-Estimated Project

16.1. Flow of an Individual Estimate on a Poorly Estimated Project

16.2. Flow of an Individual Estimate on a Well-Estimated Project

16.3. Chronological Estimation Flow for an Entire Project

Estimation Flow for Large Projects

Estimation Flow for Small Projects

16.4. Estimate Refinement

16.5. How to Present Reestimation to Other Project Stakeholders

When to Present the Reestimates

What If Your Management Won't Let You Reestimate?

16.6. A View of a Well-Estimated Project

17. Standardized Estimation Procedures

17.1. Usual Elements of a Standardized Procedure

17.2. Fitting Estimation into a Stage-Gate Process

17.3. An Example of a Standardized Estimation Procedure for Sequential Projects

17.4. An Example of a Standardized Estimation Procedure for Iterative Projects

17.5. An Example of a Standardized Estimation Procedure from an Advanced Organization

17.6. Improving Your Standardized Procedure

Additional Resources

Copyrighted material

III. Specific Estimation Challenges

18. Special Issues in Estimating Size

18.1. Challenges with Estimating Size

Role of Lines of Code in Size Estimation

18.2. Function-Point Estimation

Converting from Function Points to Lines of Code

18.3. Simplified Function-Point Techniques

The Dutch Method

GUI Elements

18.4. Summary of Techniques for Estimating Size

Additional Resources

19. Special Issues in Estimating Effort

19.1. Influences on Effort

19.2. Computing Effort from Size

Computing Effort Estimates by Using Informal Comparison to Past Projects

What Kinds of Effort Are Included in This Estimate?

19.3. Computing Effort Estimates by Using the Science of Estimation

19.4. Industry-Average Effort Graphs

19.5. ISBSG Method

Kind of project: General

Copyrighted material

Kind of project: Mainframe
Kind of project: Mid-Range
Kind of project: Desktop
Kind of project: Third Generation Language
Kind of project: Fourth Generation Language
Kind of project: Enhancement
Kind of project: New Development
19.6. Comparing Effort Estimates
Additional Resources

20. Special Issues in Estimating Schedule
20.1. The Basic Schedule Equation
20.2. Computing Schedule by Using Informal Comparisons to Past Projects
20.3. Jones's First-Order Estimation Practice
20.4. Computing a Schedule Estimate by Using the Science of Estimation
20.5. Schedule Compression and the Shortest Possible Schedule
20.6. Tradeoffs Between Schedule and Effort
Schedule Compression and Team Size
20.7. Schedule Estimation and Staffing Constraints
20.8. Comparison of Results from Different Methods
Additional Resources

Copyrighted material

21. Estimating Planning Parameters

21.1. Estimating Activity Breakdown on a Project

Estimating Allocation of Effort to Different Technical Activities

Estimating Requirements Effort

Estimating Management Effort

Estimating Total Activity

Adjustments Due to Project Type

Example of Allocating Effort to Activities

Developer-to-Tester Ratios

21.2. Estimating Schedule for Different Activities

21.3. Converting Estimated Effort (Ideal Effort) to Planned Effort

21.4. Cost Estimates

Overtime

Is the Project Cost Based on Direct Cost, Fully Burdened Cost, or Some Other Variation?

Other Direct Costs

21.5. Estimating Defect Production and Removal

Estimating Defect Removal

An Example of Estimating Defect-Removal Efficiency

21.6. Estimating Risk and Contingency Buffers

21.7. Other Rules of Thumb

21.8. Additional Resources

Copyrighted material

22. Estimate Presentation Styles

22.1. Communicating Estimate Assumptions

22.2. Expressing Uncertainty

Plus-or-Minus Qualifiers

Risk Quantification

Confidence Factors

Case-Based Estimates

Coarse Dates and Time Periods

22.3. Using Ranges (of Any Kind)

Usefulness of Estimates Presented as Ranges

Ranges and Commitments

Additional Resources

23. Politics, Negotiation, and Problem Solving

23.1. Attributes of Executives

23.2. Political Influences on Estimates

External Constraints

Budgeting and Dates

Negotiating an Estimate vs. Negotiating a Commitment

What to Do if Your Estimate Doesn't Get Accepted

Responsibility of Technical Staff to Educate Nontechnical Stakeholders

23.3. Problem Solving and Principled Negotiation

Copyrighted material

A Problem-Solving Approach to Negotiation
Separate the People from the Problem
Focus on Interests, Not Positions
Invent Options for Mutual Gain
Insist on Using Objective Criteria
Technical Staff and Technical Management Own the *Estimate*
Nontechnical Stakeholders Own the *Target*
Technical Staff and Nontechnical Staff Jointly Own the *Commitment*
Additional Resources
A. Estimate Sanity Check
Scoring
B. Answers to Chapter 2 Quiz, "Table 2-1"
C. Software Estimation Tips
Chapter 1
Chapter 2
Chapter 3
Chapter 4
Chapter 5
Chapter 6
Chapter 7
Chapter 8
Copyrighted material

Chapter 9

Chapter 10

Chapter 11

Chapter 12

Chapter 13

Chapter 14

Chapter 15

Chapter 16

Chapter 17

Chapter 18

Chapter 19

Chapter 20

Chapter 21

Chapter 22

Chapter 23

Bibliography

Steve McConnell

Index

About the Author

Copyrighted material

Welcome

The most unsuccessful three years in the education of cost estimators appears to be fifth-grade arithmetic.

—Norman R. Augustine

Software estimation is not hard. Experts have been researching and writing about software estimation for four decades, and they have developed numerous techniques that support accurate software estimates. Creating accurate estimates is straightforward—once you understand how to create them. But not all estimation practices are intuitively obvious, and even smart people won't discover all the good practices on their own. The fact that someone is an expert developer doesn't make that person an expert estimator.

Numerous aspects of estimation are not what they seem. Many so-called estimation problems arise from misunderstanding what an "estimate" is or blurring other similar-but-not-identical concepts with estimation. Some estimation practices that seem intuitively useful don't produce accurate results. Complex formulas sometimes do more harm than good, and some deceptively simple practices produce surprisingly accurate results.

This book distills four decades of research and even more decades of hands-on experience to help developers, leads, testers, and managers become effective estimators. Learning about software estimation turns out to be generally useful because the influences that affect software estimates are the influences that affect software development itself.

Art vs. Science of Software Estimation

Copyrighted material

Software estimation research is currently focused on improving estimation techniques so that sophisticated organizations can achieve project results within ±5% of estimated results instead of within ±10%. These techniques are mathematically intensive. Understanding them requires a strong math background and concentrated study. Using them requires number crunching well beyond what you can do on your hand calculator. These techniques work best when embodied in commercial software estimation tools. I refer to this set of practices as the *science of estimation*.

Meanwhile, the typical software organization is not struggling to improve its estimates from ±10% to ±5% accuracy. The typical software organization is struggling to avoid estimates that are incorrect by 100% or more. (The reasons for this are manifold and will be discussed in detail in Chapter 3 and Chapter 4.)

Our natural tendency is to believe that complex formulas like this:

$$\text{Effort} = 2.94 * (\text{KSLOC})^{[0.91 + 0.01 * \sum_{j=1}^{5} SF_j]} * \prod_{i=1}^{17} \text{EM}_i$$

will always produce more accurate results than simple formulas like this:

Effort = NumberOfRequirements * AverageEffortPerRequirement

But complex formulas aren't necessarily better. Software projects are influenced by numerous factors that undermine many of the assumptions contained in the complex formulas of the science of estimation. Those dynamics will be explained later in this book. Moreover, most software practitioners have neither the time nor the inclination to learn the intensive math required to understand the science of estimation.

Consequently, this book emphasizes rules of thumb, procedures, and simple formulas that are highly effective and understandable to practicing software professionals. These techniques will not produce estimates that are accurate to within ±5%, but they will reduce estimation error to about 25% or less, which turns out to be about as useful as most projects need, anyway. I call this set of techniques the *art of estimation*.

Copyrighted material

This book draws from both the art and science of software estimation, but its focus is on software estimation as an art.

Why This Book Was Written and Who It Is For

The literature on software estimation is widely scattered. Researchers have published hundreds of articles, and many of them are useful. But the typical practitioner doesn't have time to track down dozens of papers from obscure technical journals. A few previous books have described the science of estimation. Those books are 800–1000 pages long, require a good math background, and are targeted mainly at professional estimators—consultants or specialists who estimate large projects and do so frequently.

I wrote this book for developers, leads, testers, and managers who need to create estimates occasionally as one of their many job responsibilities. I believe that most practitioners want to improve the accuracy of their estimates but don't have the time to obtain a Ph.D. in software estimation. These practitioners struggle with practical issues like how to deal with the politics that surround the estimate, how to present an estimate so that it will actually be accepted, and how to avoid having someone change your estimate arbitrarily. If you are in this category, this book was written for you.

The techniques in this book apply to Internet and intranet development, embedded software, shrink-wrapped software, business systems, new development, legacy systems, large projects, small projects—essentially, to estimates for all kinds of software.

Key Benefits Of This Book

By focusing on the art of estimation, this book provides numerous important estimation insights:

- What an "estimate" is. (You might think you already know what an estimate is,

Copyrighted material

but common usages of the term are inaccurate in ways that undermine effective estimation.)

- The specific factors that have made your past estimates less accurate than they could have been.
- Ways to distinguish a good estimate from a poor one.
- Numerous techniques that will allow *you personally* to create good estimates.
- Several techniques you can use to help *other people on your team* create good estimates.
- Ways that *your organization* can create good estimates. (There are important differences between personal techniques, group techniques, and organizational techniques.)
- Estimation approaches that work on agile projects, and approaches that work on traditional, sequential (plan-driven) projects.
- Estimation approaches that work on small projects and approaches that work on large projects.
- How to navigate the shark-infested political waters that often surround software estimation.

In addition to gaining a better understanding of estimation concepts, the practices in this book will help you estimate numerous specific attributes of software projects, including:

- New development work, including schedule, effort, and cost
- Schedule, effort, and cost of legacy systems work
- How many features you can deliver within a specific development iteration

Copyrighted material

- The amount of functionality you can deliver for a whole project when schedule and team size are fixed
- Proportions of different software development activities needed, including how much management work, requirements, construction, testing, and defect correction will be needed
- Planning parameters, such as tradeoffs between cost and schedule, best team size, amount of contingency buffer, ratio of developers to testers, and so on
- Quality parameters, including time needed for defect correction work, defects that will remain in your software at release time, and other factors
- Practically anything else you want to estimate

In many cases, you'll be able to put this book's practices to use right away.

Most practitioners will not need to go any further than the concepts described in this book. But understanding the concepts in this book will lay enough groundwork that you'll be able to graduate to more mathematically intensive approaches later on, if you want to.

What This Book Is Not About

This book is not about how to estimate the very largest projects—more than 1 million lines of code, or more than 100 staff years. Very large projects should be estimated by professional estimators who have read the dozens of obscure journal articles, who have studied the 800–1000-page books, who are familiar with commercial estimation software, and who are skilled in both the art and science of estimation.

Where to Start

Copyrighted material

Where you start will depend on what you want to get out of the book.

If you bought this book because you need to create an estimate right now... Begin with Chapter 1, and then move to Chapter 7 and Chapter 8. After that, skim the tips in Chapter 10–Chapter 20 to find the techniques that will be the most immediately useful to you. By the way, this book's tips are highlighted in the text and numbered, and all of the tips—118 total—are also collected in Appendix C.

If you want to improve your personal estimation skills, if you want to improve your organization's estimation track record, or if you're looking for a better understanding of software estimation in general... You can read the whole book. If you like to understand general principles before you dive into the details, read the book in order. If you like to see the details first and then draw general conclusions from the details, you can start with Chapter 1, read Chapter 7 through Chapter 23, and then go back and read the earlier chapters that you skipped.

Bellevue, Washington New Year's Day, 2006

Copyrighted material

MICROSOFT PRESS SUPPORT

Every effort has been made to ensure the accuracy of this book. Microsoft Press provides corrections for books through the World Wide Web at the following address:

http://www.microsoft.com/learning/support/books/

To connect directly to the Microsoft Press Knowledge Base and enter a query regarding a question or issue that you may have, go to:

http://www.microsoft.com/mspress/support/search.asp

If you have comments, questions, or ideas regarding this book, please send them to Microsoft Press using either of the following methods:

Postal Mail:

Microsoft PressAttn: Software Estimation *EditorOne Microsoft WayRedmond, WA 98052-6399*

E-Mail:

mspinput@microsoft.com

Copyrighted material

Acknowledgments

I continue to be amazed at the many ways the Internet supports high-quality work. My first book's manuscript was reviewed almost entirely by people who lived within 50 miles of me. This book's manuscript included reviewers from Argentina, Australia, Canada, Denmark, England, Germany, Iceland, The Netherlands, Northern Ireland, Japan, Scotland, Spain, and the United States. The book has benefited enormously from these reviews.

Thanks first to the people who contributed review comments on significant portions of the book: Fernando Berzal, Steven Black, David E. Burgess, Stella M. Burns, Gavin Burrows, Dale Campbell, Robert A. Clinkenbeard, Bob Corrick, Brian Donaldson, Jason Hills, William Horn, Carl J Krzystofczyk, Jeffrey D. Moser, Thomas Oswald, Alan M. Pinder, Jon Price, Kathy Rhode, Simon Robbie, Edmund Schweppe, Gerald Simon, Creig R. Smith, Linda Taylor, and Bernd Viefhues.

Thanks also to the people who reviewed selected portions of the book: Lisa M. Adams, Hákon Ágústsson, Bryon Baker, Tina Coleman, Chris Crawford, Dominic Cronin, Jerry Deville, Conrado Estol, Eric Freeman, Hideo Fukumori, C. Dale Hildebrandt, Barbara Hitchings, Jim Holmes, Rick Hower, Kevin Hutchison, Finnur Hrafn Jonsson, Aaron Kiander, Mehmet Kerem Kiziltunç, Selimir Kustudic, Molly J. Mahai, Steve Mattingly, Joe Nicholas, Al Noel, David O'Donoghue, Sheldon Porcina, David J. Preston, Daniel Read, David Spokane, Janco Tanis, Ben Tilly, and

Copyrighted material

Wendy Wilhelm.

I'd especially like to acknowledge Construx's estimation seminar instructors. After years of stimulating discussions, it's often impossible to tell which ideas originated with me and which originated with them. Thanks to Earl Beede, Gregg Boer, Matt Peloquin, Pamela Perrott, and Steve Tockey.

This book focuses on estimation as an art, and this book's simplifications were made possible by researchers who have spent decades clarifying estimation as a science. My heartfelt appreciation to three of the giants of estimation science: Barry Boehm, Capers Jones, and Lawrence Putnam.

Working with Devon Musgrave, project editor for this book, has once again been a privilege. Thanks, Devon! Becka McKay, assistant editor, also improved my original manuscript in countless ways. Thanks also to the rest of the Microsoft Press staff, including Patricia Bradbury, Carl Diltz, Tracey Freel, Jessie Good, Patricia Masserman, Joel Panchot, and Sandi Resnick. And thanks to indexer Seth Maislin.

Thanks finally to my wife, Ashlie, who is—in my estimation—the best life partner I could ever hope for.

Copyrighted material

Equations

Copyrighted material

Equation #1
Equation #2
Equation #3
Equation #4
Equation #5
Equation #6
Equation #7
Equation #8
Equation #9
Equation #10
Equation #11
Equation #12
Equation #13
Equation #14
Equation #15
Equation #16
Equation #17
Equation #18

Copyrighted material

Figures

Figure 1-1
Figure 1-2
Figure 1-3
Figure 1-4
Figure 1-5
Figure 1-6
Figure 1-7
Figure 1-8
Figure 1-9
Figure 2-1
Figure 3-1
Figure 3-2
Figure 3-3
Figure 3-4

Copyrighted material

Figure 4-1
Figure 4-2
Figure 4-3
Figure 4-4
Figure 4-5
Figure 4-6
Figure 4-7
Figure 4-8
Figure 5-1
Figure 5-2
Figure 5-3
Figure 5-4
Figure 5-5
Figure 5-6
Figure 5-7
Figure 5-8
Figure 5-9
Figure 5-10
Figure 8-1

Copyrighted material

Figure 8-2
Figure 10-1
Figure 13-1
Figure 13-2
Figure 13-3
Figure 13-4
Figure 13-5
Figure 13-6
Figure 14-1
Figure 14-2
Figure 14-3
Figure 14-4
Figure 15-1
Figure 16-1
Figure 16-2
Figure 16-3
Figure 16-4
Figure 16-5
Figure 16-6

Copyrighted material

Figure 17-1
Figure 19-1
Figure 19-2
Figure 19-3
Figure 19-4
Figure 19-5
Figure 19-6
Figure 19-7
Figure 19-8
Figure 19-9
Figure 19-10
Figure 20-1
Figure 20-2
Figure 20-3
Figure 20-4
Figure 20-5
Table 22-1
Figure 22-2
Figure 22-3

Copyrighted material

Copyrighted material

Part I. Critical Estimation Concepts

Copyrighted material

Chapter 1. What Is an “Estimate”?

> *It is very difficult to make a vigorous, plausible, and job-risking defense of an estimate that is derived by no quantitative method, supported by little data, and certified chiefly by the hunches of the managers.*
>
> *—Fred Brooks*

You might think you already know what an estimate is. My goal by the end of this chapter is to convince you that an estimate is different from what most people think. A *good* estimate is even more different.

Here is a dictionary definition of *estimate*: 1. A tentative evaluation or rough calculation. 2. A preliminary calculation of the cost of a project. 3. A judgment based upon one’s impressions; opinion. (Source: *The American Heritage Dictionary*, Second College Edition, 1985.)

Does this sound like what you are asked for when you’re asked for an estimate? Are you asked for a *tentative* or *preliminary* calculation—that is, do you expect that you can change your mind later?

Probably not. When executives ask for an “estimate,” they’re often asking for a commitment or for a plan to meet a target. The distinctions between estimates, targets, and commitments are critical to understanding what an estimate is, what an estimate is not, and how to make your estimates better.

Copyrighted material

Estimates, Targets, and Commitments

Strictly speaking, the dictionary definition of *estimate* is correct: an estimate is a prediction of how long a project will take or how much it will cost. But estimation on software projects interplays with business targets, commitments, and control.

A *target* is a statement of a desirable business objective. Examples include the following:

- "We need to have Version 2.1 ready to demonstrate at a trade show in May."
- "We need to have this release stabilized in time for the holiday sales cycle."
- "These functions need to be completed by July 1 so that we'll be in compliance with government regulations."
- "We must limit the cost of the next release to $2 million, because that's the maximum budget we have for that release."

Businesses have important reasons to establish targets independent of software estimates. But the fact that a target is desirable or even mandatory does not necessarily mean that it is achievable.

While a target is a description of a desirable business objective, a *commitment* is a promise to deliver defined functionality at a specific level of quality by a certain date. A commitment can be the same as the estimate, or it can be more aggressive or more conservative than the estimate. In other words, do not assume that the commitment has to be the same as the estimate; it doesn't.

Copyrighted material

TIP #1

Distinguish between estimates, targets, and commitments.

Relationship Between Estimates and Plans

Estimation and planning are related topics, but estimation is not planning, and planning is not estimation. Estimation should be treated as an unbiased, analytical process; planning should be treated as a biased, goal-seeking process. With estimation, it's hazardous to want the estimate to come out to any particular answer. The goal is accuracy; the goal is not to seek a particular result. But the goal of planning is to seek a particular result. We deliberately (and appropriately) bias our plans to achieve specific outcomes. We plan specific means to reach a specific end.

Estimates form the foundation for the plans, but the plans don't have to be the same as the estimates. If the estimates are dramatically different from the targets, the project plans will need to recognize that gap and account for a high level of risk. If the estimates are close to the targets, then the plans can assume less risk.

Both estimation and planning are important, but the fundamental differences between the two activities mean that combining the two tends to lead to poor estimates *and* poor plans. The presence of a strong planning target can lead to substitution of the target for an analytically derived estimate; project members might even refer to the target as an "estimate," giving it a halo of objectivity that it doesn't deserve.

Here are examples of planning considerations that depend in part on

Copyrighted material

accurate estimates:

- Creating a detailed schedule
- Identifying a project's critical path
- Creating a complete work breakdown structure
- Prioritizing functionality for delivery
- Breaking a project into iterations

Accurate estimates support better work in each of these areas (and Chapter 21, goes into more detail on these topics).

Communicating about Estimates, Targets, and Commitments

One implication of the close and sometimes confusing relationship between estimation and planning is that project stakeholders sometimes miscommunicate about these activities. Here's an example of a typical miscommunication:

> ***Executive:*** *How long do you think this project will take? We need to have this software ready in 3 months for a trade show. I can't give you any more team members, so you'll have to do the work with your current staff. Here's a list of the features we'll need.*
>
> ***Project Lead:*** *OK, let me crunch some numbers, and get back to you.*

Later...

Copyrighted material

> ***Project Lead:*** *We've estimated the project will take 5 months.*
>
> ***Executive:*** *Five months!? Didn't you hear me? I said we needed to have this software ready in 3 months for a trade show!*

In this interaction, the project lead will probably walk away thinking that the executive is irrational, because he is asking for the team to deliver 5 months' worth of functionality in 3 months. The executive will walk away thinking that the project lead doesn't "get" the business reality, because he doesn't understand how important it is to be ready for the trade show in 3 months.

Note in this example that the executive was not really asking for an estimate; he was asking the project lead to come up with a *plan* to hit a *target*. Most executives don't have the technical background that would allow them to make fine distinctions between estimates, targets, commitments, and plans. So it becomes the technical leader's responsibility to translate the executive's request into more specific technical terms.

Here's a more productive way that the interaction could go:

Copyrighted material

Executive: *How long do you think this project will take? We need to have this software ready in 3 months for a trade show. I can't give you any more team members, so you'll have to do the work with your current staff. Here's a list of the features we'll need.*

Project Lead: *Let me make sure I understand what you're asking for. Is it more important for us to deliver 100% of these features, or is it more important to have something ready for the trade show?*

Executive: *We have to have something ready for the trade show. We'd like to have 100% of those features if possible.*

Project Lead: *I want to be sure I follow through on your priorities as best I can. If it turns out that we can't deliver 100% of the features by the trade show, should we be ready to ship what we've got at trade show time, or should we plan to slip the ship date beyond the trade show?*

Executive: *We have to have something for the trade show, so if push comes to shove, we have to ship something, even if it isn't 100% of what we want.*

Project Lead: *OK, I'll come up with a plan for delivering as many features as we can in the next 3 months.*

> **TIP #2**
>
> When you're asked to provide an estimate, determine whether you're supposed to be estimating or figuring out how to hit a target.

Estimates as Probability Statements

If three-quarters of software projects overrun their estimates, the odds of any given software project completing on time and within budget are not 100%. Once we recognize that the odds of on-time completion are not

Copyrighted material

100%, an obvious question arises: “If the odds aren’t 100%, what are they?” This is one of the central questions of software estimation.

Software estimates are routinely presented as single-point numbers, such as “This project will take 14 weeks.” Such simplistic single-point estimates are meaningless because they don’t include any indication of the probability associated with the single point. They imply a probability as shown in Figure 1-1—the only possible outcome is the single point given.

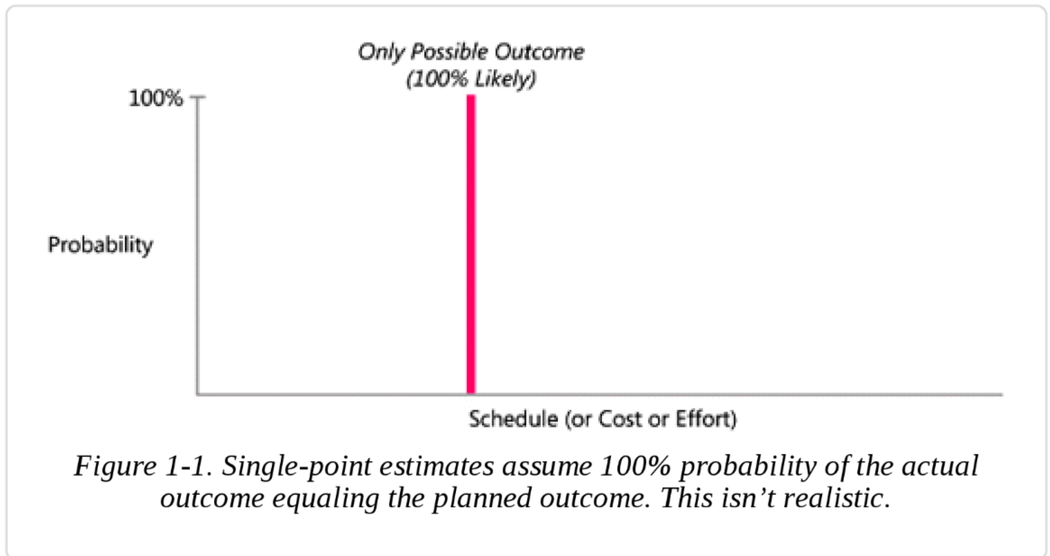


Figure 1-1. Single-point estimates assume 100% probability of the actual outcome equaling the planned outcome. This isn’t realistic.

A single-point estimate is usually a target masquerading as an estimate. Occasionally, it is the sign of a more sophisticated estimate that has been stripped of meaningful probability information somewhere along the way.

Copyrighted material

TIP #3

When you see a single-point "estimate," ask whether the number is an estimate or whether it's really a target.

Accurate software estimates acknowledge that software projects are assailed by uncertainty from all quarters. Collectively, these various sources of uncertainty mean that project outcomes follow a probability distribution—some outcomes are more likely, some outcomes are less likely, and a cluster of outcomes in the middle of the distribution are most likely. You might expect that the distribution of project outcomes would look like a common bell curve, as shown in Figure 1-2.

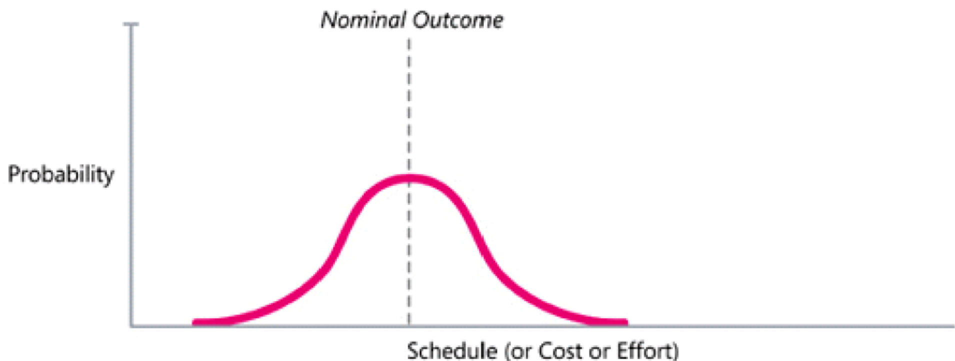


Figure 1-2. A common assumption is that software project outcomes follow a bell curve. This assumption is incorrect because there are limits to how efficiently a project team can complete any given amount of work.

Copyrighted material

Each point on the curve represents the chance of the project finishing exactly on that date (or costing exactly that much). The area under the curve adds up to 100%. This sort of probability distribution acknowledges the possibility of a broad range of outcomes. But the assumption that the outcomes are symmetrically distributed about the mean (average) is not valid. There is a limit to how well a project can be conducted, which means that the tail on the left side of the distribution is truncated rather than extending as far to the left as it does in the bell curve. And while there is a limit to how well a project can go, there is no limit to how poorly a project can go, and so the probability distribution does have a very long tail on the right.

Figure 1-3 provides an accurate representation of the probability distribution of a software project's outcomes.

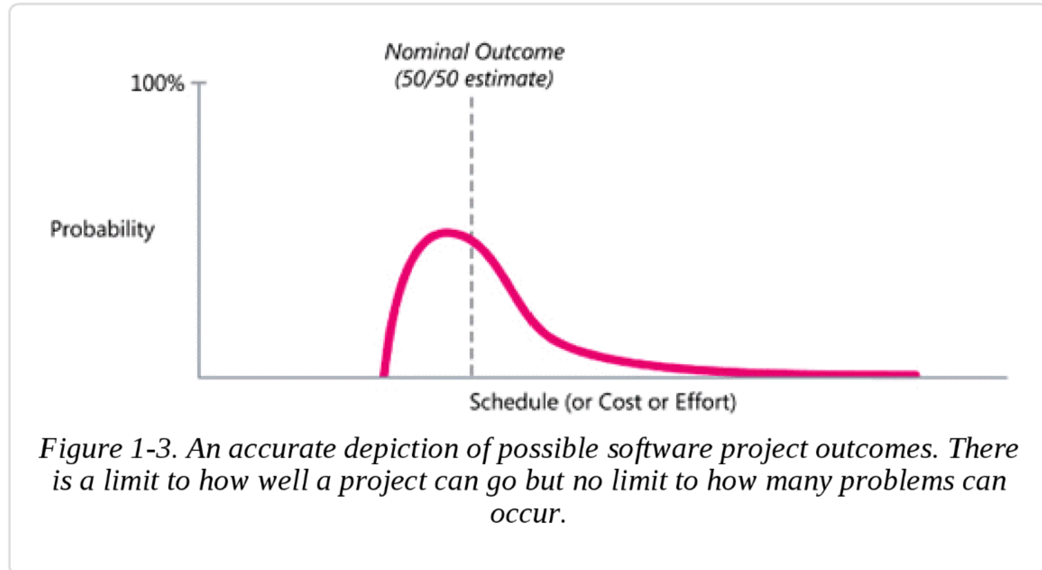


Figure 1-3. An accurate depiction of possible software project outcomes. There is a limit to how well a project can go but no limit to how many problems can occur.

Copyrighted material

The vertical dashed line shows the “nominal” outcome, which is also the “50/50” outcome—there’s a 50% chance that the project will finish better and a 50% chance that it will finish worse. Statistically, this is known as the “median” outcome.

Figure 1-4 shows another way of expressing this probability distribution. While Figure 1-3 showed the probabilities of delivering on specific dates, Figure 1-5 shows the probabilities of delivering on each specific date *or earlier*.

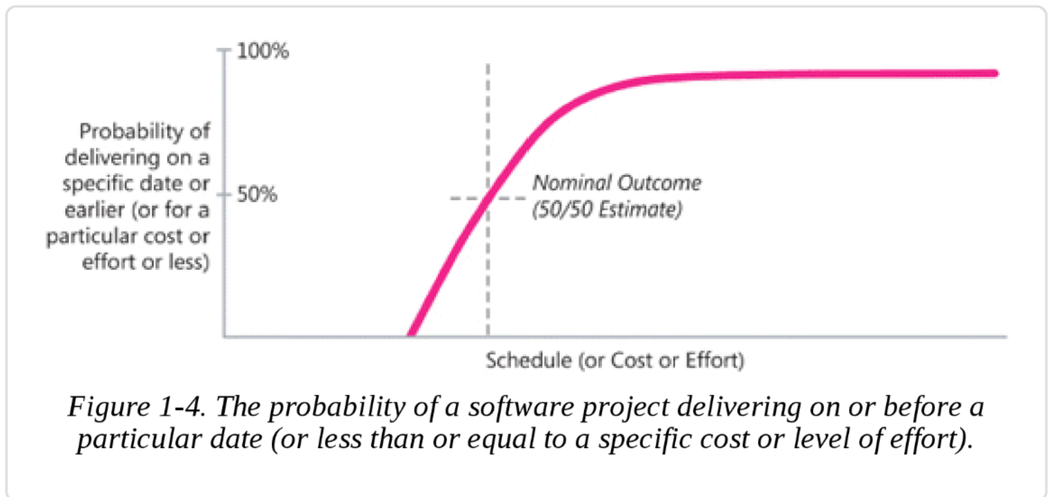


Figure 1-4. The probability of a software project delivering on or before a particular date (or less than or equal to a specific cost or level of effort).

Copyrighted material

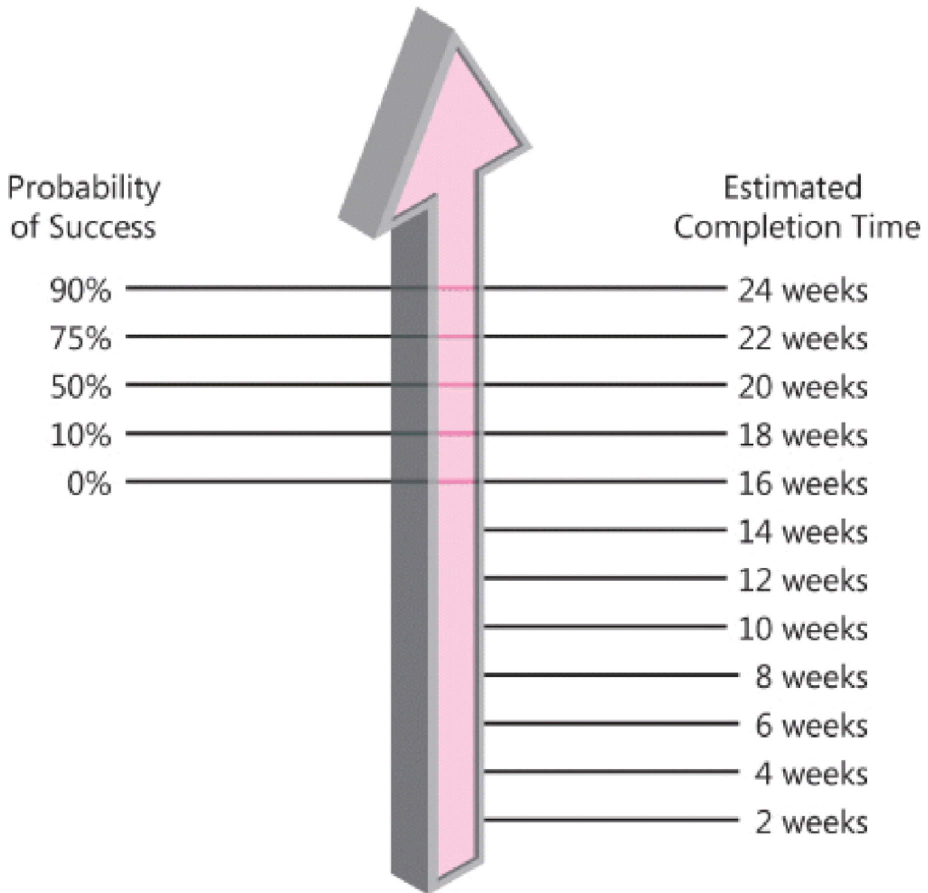


Figure 1-5. All single-point estimates are associated with a probability, explicitly or implicitly.

Figure 1-5 presents the idea of probabilistic project outcomes in another way. As you can see from the figure, a naked estimate like "18 weeks" leaves out the interesting information that 18 weeks is only 10% likely. An

Copyrighted material

estimate like "18 to 24 weeks" is more informative and conveys useful information about the likely range of project outcomes.

> **TIP #4**
>
> When you see a single-point estimate, that number's probability is not 100%. Ask what the probability of that number is.

You can express probabilities associated with estimates in numerous ways. You could use a "percent confident" attached to a single-point number: "We're 90% confident in the 24-week schedule." You could describe estimates as best case and worst case, which implies a probability: "We estimate a best case of 18 weeks and a worst case of 24 weeks." Or you could simply state the estimated outcome as a range rather than a single-point number: "We're estimating 18 to 24 weeks." The key point is that all estimates include a probability, whether the probability is stated or implied. An explicitly stated probability is one sign of a good estimate.

You can make a commitment to the optimistic end or the pessimistic end of an estimation range—or anywhere in the middle. The important thing is for you to know where in the range your commitment falls so that you can plan accordingly.

Common Definitions of a "Good" Estimate

The answer to the question of what an "estimate" is still leaves us with the question of what a *good* estimate is. Estimation experts have proposed

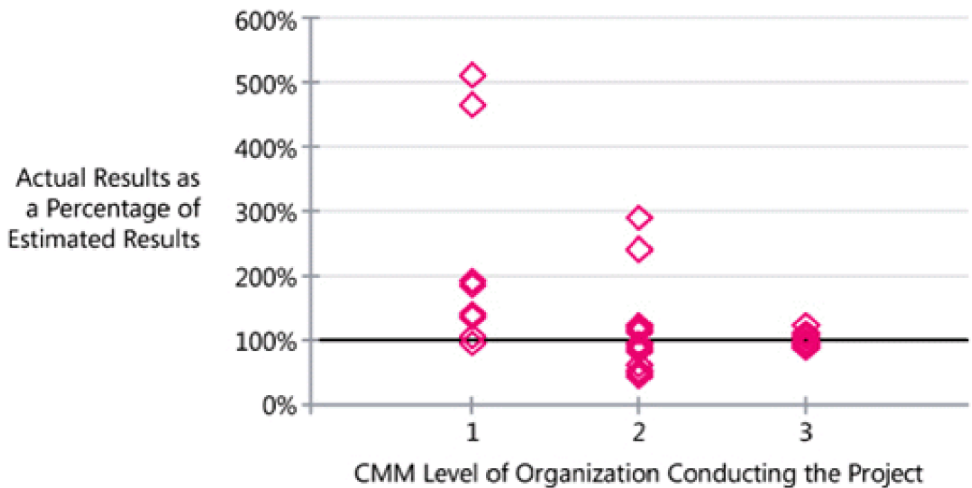
Copyrighted material

various definitions of a good estimate. Capers Jones has stated that accuracy with ±10% is possible, but only on well-controlled projects (Jones 1998). Chaotic projects have too much variability to achieve that level of accuracy.

In 1986, Professors S.D. Conte, H.E. Dunsmore, and V.Y. Shen proposed that a good estimation approach should provide estimates that are within 25% of the actual results 75% of the time (Conte, Dunsmore, and Shen 1986). This evaluation standard is the most common standard used to evaluate estimation accuracy (Stutzke 2005).

Numerous companies have reported estimation results that are close to the accuracy Conte, Dunsmore, and Shen and Jones have suggested. Figure 1-6 shows actual results compared to estimates from a set of U.S. Air Force projects.

Copyrighted material



Source: “A Correlational Study of the CMM and Software Development Performance” (Lawlis, Flowe, and Thordahl 1995).

Figure 1-6. Improvement in estimation of a set of U.S. Air Force projects. The predictability of the projects improved dramatically as the organizations moved toward higher CMM levels.[1]

Figure 1-7 shows results of a similar improvement program at the Boeing Company.

Copyrighted material

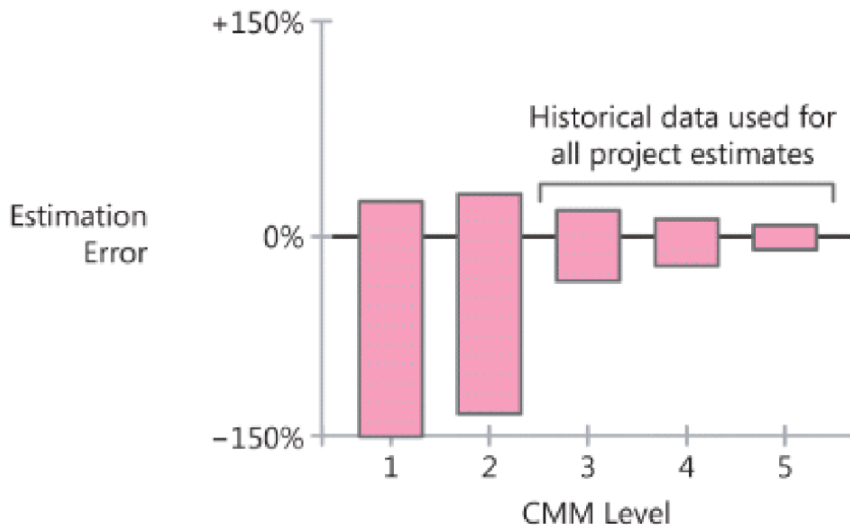


Figure 1-7. Improvement in estimation at the Boeing Company. As with the U.S. Air Force projects, the predictability of the projects improved dramatically at higher CMM levels.

A final, similar example, shown in Figure 1-8, comes from improved estimation results at Schlumberger.

Copyrighted material

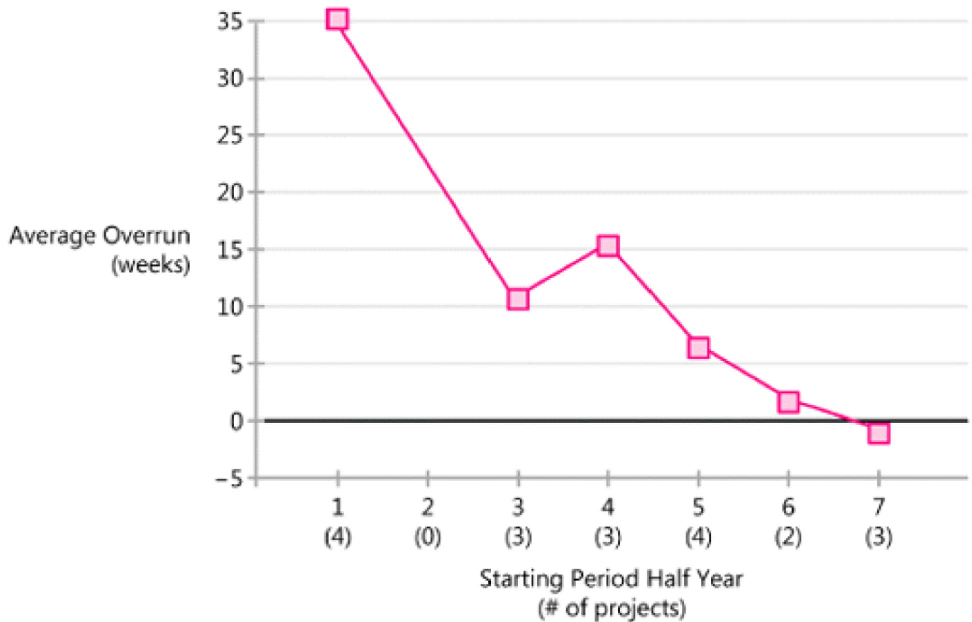


Figure 1-8. Schlumberger improved its estimation accuracy from an average overrun of 35 weeks to an average underrun of 1 week.

One of my client companies delivers 97% of its projects on time and within budget. Telcordia reported that it delivers 98% of its projects on time and within budget (Pitterman 2000). Numerous other companies have published similar results (Putnam and Myers 2003). Organizations are creating good estimates by both Jones's definition and Conte, Dunsmore, and Shen's definition. However, an important concept is missing from both of these definitions—namely, that accurate estimation results cannot be accomplished through estimation practices alone. They must also be supported by effective project control.

Copyrighted material

Estimates and Project Control

Sometimes when people discuss software estimation they treat estimation as a purely predictive activity. They act as though the estimate is made by an impartial estimator, sitting somewhere in outer space, disconnected from project planning and prioritization activities.

In reality, there is little that is pure about software estimation. If you ever wanted an example of Heisenberg's Uncertainty Principle applied to software, estimation would be it. (Heisenberg's Uncertainty Principle is the idea that the mere act of observing a thing changes it, so you can never be sure how that thing would behave if you weren't observing it.) Once we make an estimate and, on the basis of that estimate, make a commitment to deliver functionality and quality by a particular date, then we *control* the project to meet the target. Typical project control activities include removing noncritical requirements, redefining requirements, replacing less-experienced staff with more-experienced staff, and so on. Figure 1-9 illustrates these dynamics.

Copyrighted material

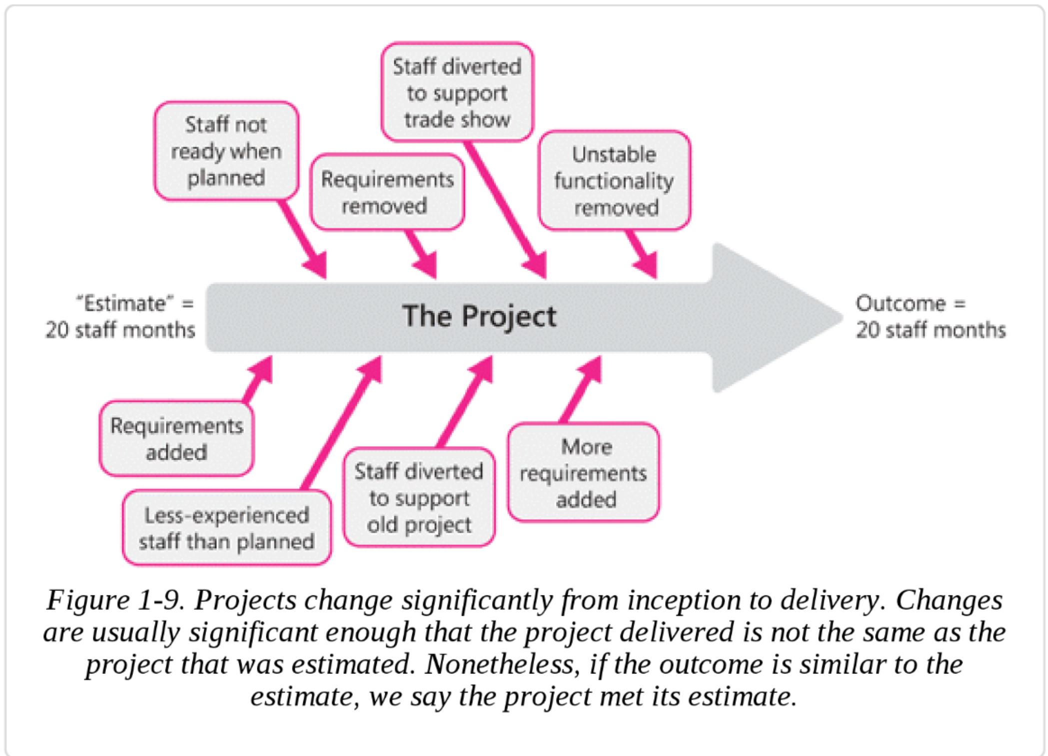


Figure 1-9. Projects change significantly from inception to delivery. Changes are usually significant enough that the project delivered is not the same as the project that was estimated. Nonetheless, if the outcome is similar to the estimate, we say the project met its estimate.

In addition to project control activities, projects are often affected by unforeseen external events. The project team might need to create an interim release to support a key customer. Staff might be diverted to support an old project, and so on.

Events that happen during the project nearly always invalidate the assumptions that were used to estimate the project in the first place. Functionality assumptions change, staffing assumptions change, and priorities change. It becomes impossible to make a clean analytical assessment of whether the project was estimated accurately—because the software project that was ultimately delivered is not the project that was

Copyrighted material

originally estimated.

In practice, if we deliver a project with about the level of functionality intended, using about the level of resources planned, in about the time frame targeted, then we typically say that the project "met its estimates," despite all the analytical impurities implicit in that statement.

Thus, the criteria for a "good" estimate cannot be based on its predictive capability, which is impossible to assess, but on the estimate's ability to support project success, which brings us to the next topic: the Proper Role of Estimation.

Estimation's Real Purpose

Suppose you're preparing for a trip and deciding which suitcase to take. You have a small suitcase that you like because it's easy to carry and will fit into an airplane's overhead storage bin. You also have a large suitcase, which you don't like because you'll have to check it in and then wait for it at baggage claim, lengthening your trip. You lay your clothes beside the small suitcase, and it appears that they will almost fit. What do you do? You might try packing them very carefully, not wasting any space, and hoping they all fit. If that approach doesn't work, you might try stuffing them into the suitcase with brute force, sitting on the top and trying to squeeze the latches closed. If that still doesn't work, you're faced with a choice: leave a few clothes at home or take the larger suitcase.

Software projects face a similar dilemma. Project planners often find a gap between a project's business targets and its estimated schedule and cost. If the gap is small, the planner might be able to control the project to a successful conclusion by preparing extra carefully or by squeezing the

Copyrighted material

project's schedule, budget, or feature set. If the gap is large, the project's targets must be reconsidered.

The primary purpose of software estimation is not to predict a project's outcome; it is to determine whether a project's targets are realistic enough to allow the project to be controlled to meet them. Will the clothes you want to take on your trip fit into the small suitcase or will you be forced to take the large suitcase? Can you take the small suitcase if you make minor adjustments? Executives want the same kinds of answers. They often don't want an accurate estimate that tells them that the desired clothes won't fit into the suitcase; they want a plan for making as many of the clothes fit as possible.

Problems arise when the gap between the business targets and the schedule and effort needed to achieve those targets becomes too large. I have found that if the initial target and initial estimate are within about 20% of each other, the project manager will have enough maneuvering room to control the feature set, schedule, team size, and other parameters to meet the project's business goals; other experts concur (Boehm 1981, Stutzke 2005). If the gap between the target and what is actually needed is too large, the manager will not be able to control the project to a successful conclusion by making minor adjustments to project parameters. No amount of careful packing or sitting on the suitcase will squeeze all your clothes into the smaller suitcase, and you'll have to take the larger one, even if it isn't your first choice, or you'll have to leave some clothes behind. The project targets will need to be brought into better alignment with reality before the manager can control the project to meet its targets.

Estimates don't need to be perfectly accurate as much as they need to be *useful*. When we have the combination of accurate estimates, good target setting, and good planning and control, we can end up with project results

Copyrighted material

that are close to the "estimates." (As you've guessed, the word "estimate" is in quotation marks because the project that was estimated is not the same project that was ultimately delivered.)

These dynamics of changing project assumptions are a major reason that this book focuses more on the art of estimation than on the science. Accuracy of ±5% won't do you much good if the project's underlying assumptions change by 100%.

A Working Definition of a "Good Estimate"

With the background provided in the past few sections, we're now ready to answer the question of what qualifies as a good estimate.

> *A good estimate is an estimate that provides a clear enough view of the project reality to allow the project leadership to make good decisions about how to control the project to hit its targets..*

This definition is the foundation of the estimation discussion throughout the rest of this book.

Additional Resources

[biblio01_001] Conte,S.D., H.E.Dunsmore, and V.Y.Shen. *Software Engineering Metrics and Models*. Menlo Park, CA: Benjamin/Cummings, 1986. Conte, Dunsmore, and Shen's book contains the definitive discussion of evaluating estimation models. It discusses the "within 25% of actual 75% of the time" criteria, as well as many other evaluation criteria.

Copyrighted material

[biblio01_002] DeMarco,Tom. *Controlling Software Projects*. New York, NY: Yourdon Press, 1982. DeMarco discusses the probabilistic nature of software projects.

[biblio01_003] Stutzke,RichardD. *Estimating Software-Intensive Systems*. Upper Saddle River, NJ: Addison-Wesley, 2005. Appendix C of Stutzke's book contains a summary of measures of estimation accuracy.

---

[1] The CMM (Capability Maturity Model) is a system defined by the Software Engineering Institute to assess the effectiveness of software organizations.

Copyrighted material

Chapter 2. How Good an Estimator Are You?

> *The process is called estimation, not exactimation.*
>
> *—Phillip Armour*

Now that you know what a good estimate is, how good an estimator are you? The following section will help you find out.

A Simple Estimation Quiz

Table 2-1, appearing on the following page, contains a quiz designed to test your estimation skills. Please read and observe the following directions carefully:

Table 2-1. How Good an Estimator Are You?

[Low Estimate – High Estimate]	Description
[________________ – ________________]	Surface temperature of the Sun
[________________ – ________________]	Latitude of Shanghai

Copyrighted material

[________________ – ________________]	Area of the Asian continent
[________________ – ________________]	The year of Alexander the Great's birth
[________________ – ________________]	Total value of U.S. currency in circulation in 2004
[________________ – ________________]	Total volume of the Great Lakes
[________________ – ________________]	Worldwide box office receipts for the movie *Titanic*
[________________ – ________________]	Total length of the coastline of the Pacific Ocean
[________________ – ________________]	Number of book titles published in the U.S. since 1776
[________________ – ________________]	Heaviest blue whale ever recorded

Source: Inspired by a similar quiz in *Programming Pearls*, Second Edition (Bentley 2000).

This quiz is from *Software Estimation* by Steve McConnell (Microsoft Press, 2006) and is © 2006 Steve McConnell. All Rights Reserved. Permission to copy this quiz is granted provided that this copyright notice is included.

For each question, fill in the upper and lower bounds that, in your opinion, give you a 90% chance of including the correct value. Be careful not to make your ranges either too wide or too narrow. Make them wide enough so that, in your best judgment, the ranges give you a 90% chance of

Copyrighted material

including the correct answer. Please do not research any of the answers—this quiz is intended to assess your estimation skills, not your research skills. You must fill in an answer for each item; an omitted item will be scored as an incorrect item. Please limit your time on this exercise to 10 minutes.

(Also, you might want to photocopy the quiz before taking it so that the next person who reads this book can take it, too.)

The correct answers to this exercise (the latitude of Shanghai, for example) are listed in Appendix B in the back of the book. Give yourself one point for each of your ranges that includes the related correct answer.

How did you do? (Don't feel bad. Most people do poorly on this quiz!) Please write your score here: _______________

Discussion of Quiz Results

The purpose of this quiz is not to determine whether you know when Alexander the Great was born or the latitude of Shanghai. Its purpose is to determine how well you understand your own estimation capabilities.

How Confident Is "90% Confident"?

The directions above are specific that the goal of the exercise is to estimate at the 90% confidence level. Because there are 10 questions in the quiz, if you were truly estimating at the 90% confidence level, you should have gotten about 9 answers correct.[1]

If you were cautious, you made your ranges conservatively wide, in which case you scored 10 out of 10 correctly. If you were just a little hasty, you

Copyrighted material

made your ranges narrower than they needed to be, in which case you scored 7 or 8 out of 10 correctly. I've given this quiz to hundreds of estimators. Figure 2-1 shows the results from the most recent 600 people who have taken the quiz.

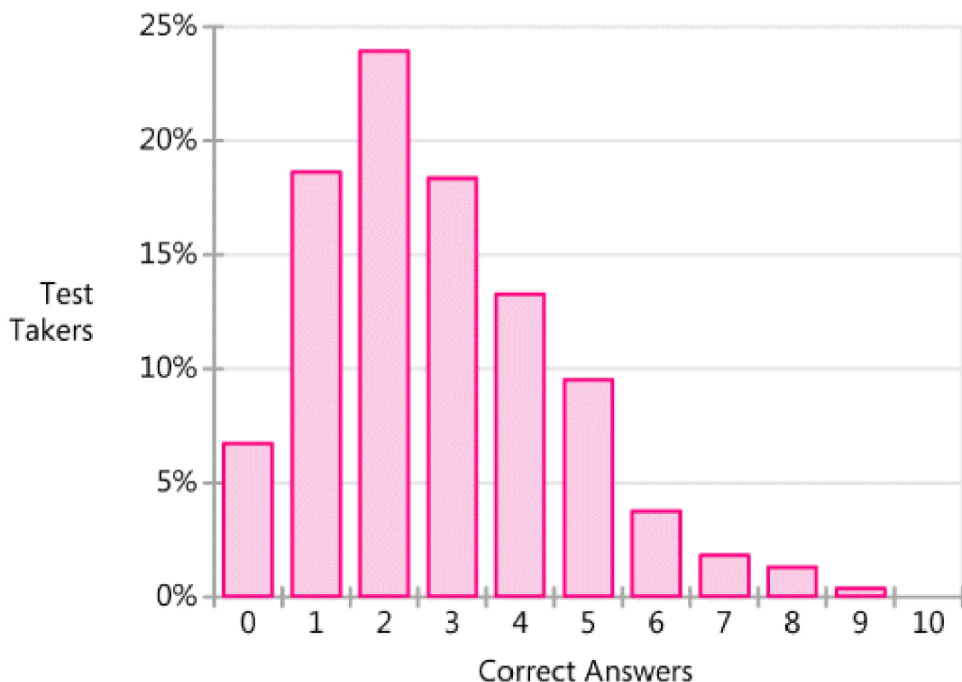


Figure 2-1. Results from administering the "How Good an Estimator Are You?" quiz. Most quiz-takers get 1–3 answers correct.

For the test takers whose results are shown in the figure, the average number of correct answers is 2.8. Only 2 percent of quiz takers score 8 or

Copyrighted material

more answers correctly. No one has ever gotten 10 correct. I've concluded that most people's intuitive sense of "90% confident" is really comparable to something closer to "30% confident." Other studies have confirmed this basic finding (Zultner 1999, Jørgensen 2002).

Similarly, I've seen numerous project teams present "90% confident" schedules, and I've frequently seen those project teams overrun their "90% confident" schedules—more often than not. If those schedules represented a true 90% confidence, the project teams would overrun them only about 1 time out of 10.

I've concluded that specific percentages such as "90%" are not meaningful unless they are supported through some kind of statistical analysis. Otherwise those specific percentages are just wishful thinking. How to get to a *real* 90% confidence level will be discussed later in the book.

TIP #5

Don't provide "percentage confident" estimates (especially "90% confident") unless you have a quantitatively derived basis for doing so.

If you didn't take the quiz earlier in this chapter, this would be a good time to go back and take it. I think you'll be surprised at how few answers you get correct even after reading this explanation.

How Wide Should You Make Your Ranges?

When I find the rare person who gets 7 or 8 answers correct, I ask "How

Copyrighted material

did you get that many correct?” The typical response? “I made my ranges too wide.”

My response is, “No, you didn’t! You didn’t make your ranges wide enough!” If you got only 7 or 8 correct, your ranges were still too narrow to include the correct answer as often as you should have.

We are conditioned to believe that estimates expressed as narrow ranges are more accurate than estimates expressed as wider ranges. We believe that wide ranges make us appear ignorant or incompetent. The opposite is usually the case. (Of course, narrow ranges are desirable in the cases when the underlying data supports them.)

> **TIP #6**
>
> Avoid using artificially narrow ranges. Be sure the ranges you use in your estimates don’t misrepresent your confidence in your estimates.

Where Does Pressure to Use Narrow Ranges Come From?

When you were taking the quiz, did you feel pressure to make your ranges wider? Or did you feel pressure to make your ranges narrower? Most people report that they feel pressure to make the ranges as narrow as possible. But if you go back and review the instructions, you’ll find that they do not encourage you to use narrow ranges. Indeed, I was careful to state that you should make your ranges neither too wide nor too narrow—just wide enough to give you a 90% confidence in including the correct answer.

Copyrighted material

After discussing this issue with hundreds of developers and managers, I've concluded that much of the pressure to provide narrow ranges is self-induced. Some of the pressure comes from people's sense of professional pride. They believe that narrow ranges are a sign of a better estimate, even though that isn't the case. And some of the pressure comes from experiences with bosses or customers who insisted on the use of overly narrow ranges.

This same self-induced pressure has been found in interactions between customers and estimators. Jørgensen and Sjøberg reported that information about customers' expectations exerts strong influence on estimates and that estimators are typically not conscious of the degree to which their estimates are affected (Jørgensen and Sjøberg 2002).

TIP #7

If you are feeling pressure to make your ranges narrower, verify that the pressure actually is coming from an external source and not from yourself.

For those cases in which the pressure truly is coming from an external source, Chapter 22, and Chapter 23. discuss how to deal with that pressure.

How Representative Is This Quiz of Real Software Estimates?

In software, you aren't often asked to estimate the volume of the Great Lakes or the surface temperature of the Sun. Is it reasonable to expect you

Copyrighted material

to be able to estimate the amount of U.S. currency in circulation or the number of books published in the U.S., especially if you're not in the U.S.?

Software developers are often asked to estimate projects in unfamiliar business areas, projects that will be implemented in new technologies, the impacts of new programming tools on productivity, the productivity of unidentified personnel, and so on. Estimating in the face of uncertainty is business as usual for software estimators. The rest of this book explains how to succeed in such circumstances.

---

[1] The mathematics behind "90% confident" are a little complicated. If you were really estimating with 90% confidence, you would have a 34.9% chance of getting 10 answers correct, 38.7% chance of getting 9 answers correct, and a 19.4% chance of getting 8 answers correct. In other words, you'd have a 93% chance of getting 8 or more answers correct.

Copyrighted material

Chapter 3. Value of Accurate Estimates

> *[The common definition of estimate is] "the most optimistic prediction that has a non-zero probability of coming true." ... Accepting this definition leads irrevocably toward a method called what's-the-earliest-date-by-which-you-can't-prove-you-won't-be-finished estimating.*
>
> *—Tom DeMarco*

The inaccuracy of software project estimates—as muddied by unrealistic targets and unachievable commitments—has been a problem for many years. In the 1970s, Fred Brooks pointed out that "more software projects have gone awry for lack of calendar time than all other causes combined" (Brooks 1975). A decade later, Scott Costello observed that "deadline pressure is the single greatest enemy of software engineering" (Costello 1984). In the 1990s, Capers Jones reported that "excessive or irrational schedules are probably the single most destructive influence in all of software" (Jones 1994, 1997).

Tom DeMarco wrote his common definition of an estimate in 1982. Despite the successes I mentioned in the first chapter, not much has changed in the years since he wrote that definition. You might already agree that accurate estimates are valuable. This chapter details the specific benefits of accurate estimates and provides supporting data for them.

Copyrighted material

Is It Better to Overestimate or Underestimate?

Intuitively, a perfectly accurate estimate forms the ideal planning foundation for a project. If the estimates are accurate, work among different developers can be coordinated efficiently. Deliveries from one development group to another can be planned to the day, hour, or minute. We know that accurate estimates are rare, so if we're going to err, is it better to err on the side of overestimation or underestimation?

Arguments Against Overestimation

Managers and other project stakeholders sometimes fear that, if a project is overestimated, Parkinson's Law will kick in—the idea that work will expand to fill available time. If you give a developer 5 days to deliver a task that could be completed in 4 days, the developer will find something to do with the extra day. If you give a project team 6 months to complete a project that could be completed in 4 months, the project team will find a way to use up the extra 2 months. As a result, some managers consciously squeeze the estimates to try to avoid Parkinson's Law.

Another concern is Goldratt's "Student Syndrome" (Goldratt 1997). If developers are given too much time, they'll procrastinate until late in the project, at which point they'll rush to complete their work, and they probably won't finish the project on time.

A related motivation for underestimation is the desire to instill a sense of urgency in the development team. The line of reason goes like this:

Copyrighted material

The developers say that this project will take 6 months. I think there's some padding in their estimates and some fat that can be squeezed out of them. In addition, I'd like to have some schedule urgency on this project to force prioritizations among features. So I'm going to insist on a 3-month schedule. I don't really believe the project can be completed in 3 months, but that's what I'm going to present to the developers. If I'm right, the developers might deliver in 4 or 5 months. Worst case, the developers will deliver in the 6 months they originally estimated.

Are these arguments compelling? To determine that, we need to examine the arguments in favor of erring on the side of overestimation.

Arguments Against Underestimation

Underestimation creates numerous problems—some obvious, some not so obvious.

Reduced effectiveness of project plans. Low estimates undermine effective planning by feeding bad assumptions into plans for specific activities. They can cause planning errors in the team size, such as planning to use a team that's smaller than it should be. They can undermine the ability to coordinate among groups—if the groups aren't ready when they said they would be, other groups won't be able to integrate with their work.

If the estimation errors caused the plans to be off by only 5% or 10%, those errors wouldn't cause any significant problems. But numerous studies have found that software estimates are often inaccurate by 100% or more (Lawlis, Flowe, and Thordahl 1995; Jones 1998; Standish Group 2004; ISBSG 2005). When the planning assumptions are wrong by this magnitude, the average project's plans are based on assumptions that are so far off that the plans are virtually useless.

Statistically reduced chance of on-time completion. Developers typically

Copyrighted material

estimate 20% to 30% lower than their actual effort (van Genuchten 1991). Merely using their normal estimates makes the project plans optimistic. Reducing their estimates even further simply reduces the chances of on-time completion even more.

Poor technical foundation leads to worse-than-nominal results. A low estimate can cause you to spend too little time on upstream activities such as requirements and design. If you don't put enough focus on requirements and design, you'll get to redo your requirements and redo your design later in the project—at greater cost than if you'd done those activities well in the first place (Boehm and Turner 2004, McConnell 2004a). This ultimately makes your project take longer than it would have taken with an accurate estimate.

Destructive late-project dynamics make the project worse than nominal. Once a project gets into "late" status, project teams engage in numerous activities that they don't need to engage in during an "on-time" project. Here are some examples:

- More status meetings with upper management to discuss how to get the project back on track.
- Frequent reestimation, late in the project, to determine just when the project will be completed.
- Apologizing to key customers for missing delivery dates (including attending meetings with those customers).
- Preparing interim releases to support customer demos, trade shows, and so on. If the software were ready on time, the software itself could be used, and no interim release would be necessary.

Copyrighted material

- More discussions about which requirements absolutely must be added because the project has been underway so long.
- Fixing problems arising from quick and dirty workarounds that were implemented earlier in response to the schedule pressure.

The important characteristic of each of these activities is that they don't need to occur *at all* when a project is meeting its goals. These extra activities drain time away from productive work on the project and make it take longer than it would if it were estimated and planned accurately.

Weighing the Arguments

Goldratt's Student Syndrome can be a factor on software projects, but I've found that the most effective way to address Student Syndrome is through active task tracking and buffer management (that is, project control), similar to what Goldratt suggests, not through biasing the estimates.

As Figure 3-1 shows, the best project results come from the most accurate estimates (Symons 1991). If the estimate is too low, planning inefficiencies will drive up the actual cost and schedule of the project. If the estimate is too high, Parkinson's Law kicks in.

Copyrighted material

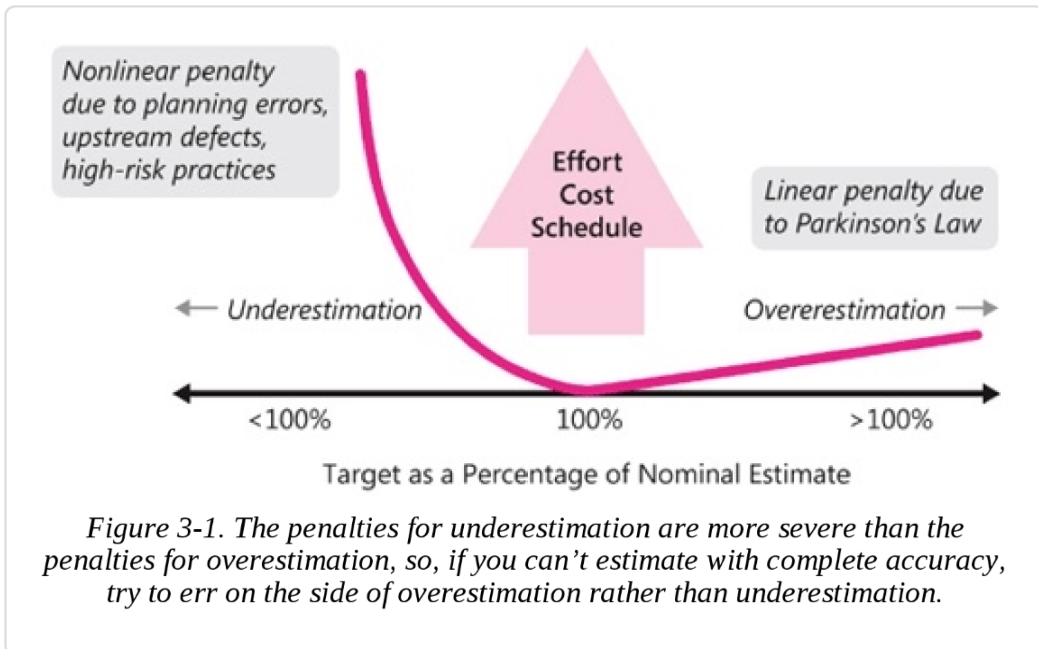


Figure 3-1. The penalties for underestimation are more severe than the penalties for overestimation, so, if you can't estimate with complete accuracy, try to err on the side of overestimation rather than underestimation.

I believe that Parkinson's Law does apply to software projects. Work does expand to fill available time. But deliberately underestimating a project because of Parkinson's Law makes sense only if the penalty for overestimation is worse than the penalty for underestimation. In software, the penalty for overestimation is *linear and bounded*—work will expand to fill available time, but it will not expand any further. But the penalty for underestimation is *nonlinear and unbounded*—planning errors, shortchanging upstream activities, and the creation of more defects cause more damage than overestimation does, and with little ability to predict the extent of the damage ahead of time.

Copyrighted material

TIP #8

Don't intentionally underestimate. The penalty for underestimation is more severe than the penalty for overestimation. Address concerns about overestimation through planning and control, not by biasing your estimates.

Details on the Software Industry's Estimation Track Record

The software industry's estimation track record provides some interesting clues to the nature of software's estimation problems. In recent years, The Standish Group has published a biennial survey called "The Chaos Report," which describes software project outcomes. In the 2004 report, 54% of projects were delivered late, 18% failed outright, and 28% were delivered on time and within budget. Figure 3-2 shows the results for the 10 years from 1994 to 2004.

Copyrighted material

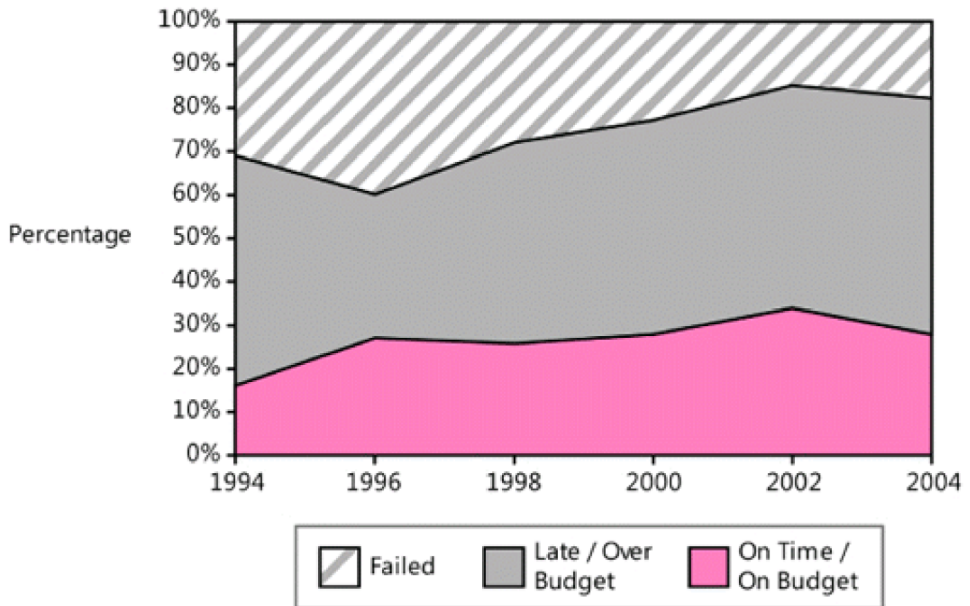


Figure 3-2. Project outcomes reported in The Standish Group's Chaos report have fluctuated year to year. About three quarters of all software projects are delivered late or fail outright.

What's notable about The Standish Group's data is that it doesn't even have a category for early delivery! The best possible performance is meeting expectations "On Time/On Budget"—and the other options are all downhill from there.

Capers Jones presents another view of project outcomes. Jones has observed for many years that project success depends on project size. That is, larger projects struggle more than smaller projects do. Table 3-1 illustrates this point.

Copyrighted material

Table 3-1. Project Outcomes by Project Size

Size in Function Points (and Approximate Lines of Code)	Early	On Time	Late	Failed (Canceled)
10 FP (1,000 LOC)	11%	81%	6%	2%
100 FP (10,000 LOC)	6%	75%	12%	7%
1,000 FP (100,000 LOC)	1%	61%	18%	20%
10,000 FP (1,000,000 LOC)	<1%	28%	24%	48%
100,000 FP (10,000,000 LOC)	0%	14%	21%	65%

Source: *Estimating Software Costs* (Jones 1998).

As you can see from Jones's data, the larger a project, the less chance the project has of completing on time and the greater chance it has of failing outright.

Overall, a compelling number of studies have found results in line with the results reported by The Standish Group and Jones, that about one quarter of all projects are delivered on time; about one quarter are canceled; and about half are delivered late, over budget, or both (Lederer and Prasad 1992; Jones 1998; ISBSG 2001; Krasner 2003; Putnam and Myers 2003; Heemstra, Siskens and van der Stelt 2003; Standish Group 2004).

The reasons that projects miss their targets are manifold. Poor estimates are one reason but not the only reason. We'll discuss the reasons in depth in Chapter 4.

Copyrighted material

How Late Are the Late Projects?

The number of projects that run late or over budget is one consideration. The degree to which these projects miss their targets is another consideration. According to the first Standish Group survey, the average project schedule overrun was about 120% and the average cost overrun was about 100% (Standish Group 1994). But the estimation accuracy is probably worse than those numbers reflect. The Standish Group found that late projects routinely threw out significant amounts of functionality to achieve the schedules and budgets they eventually did meet. Of course, these projects' estimates weren't for the abbreviated versions they eventually delivered; they were for the originally specified, full-featured versions. If these late projects had delivered all of their originally specified functionality, they would have overrun their plans even more.

One Company's Experience

A more company-specific view of project outcomes is shown in the data reported by one of my clients in Figure 3-3.

Copyrighted material

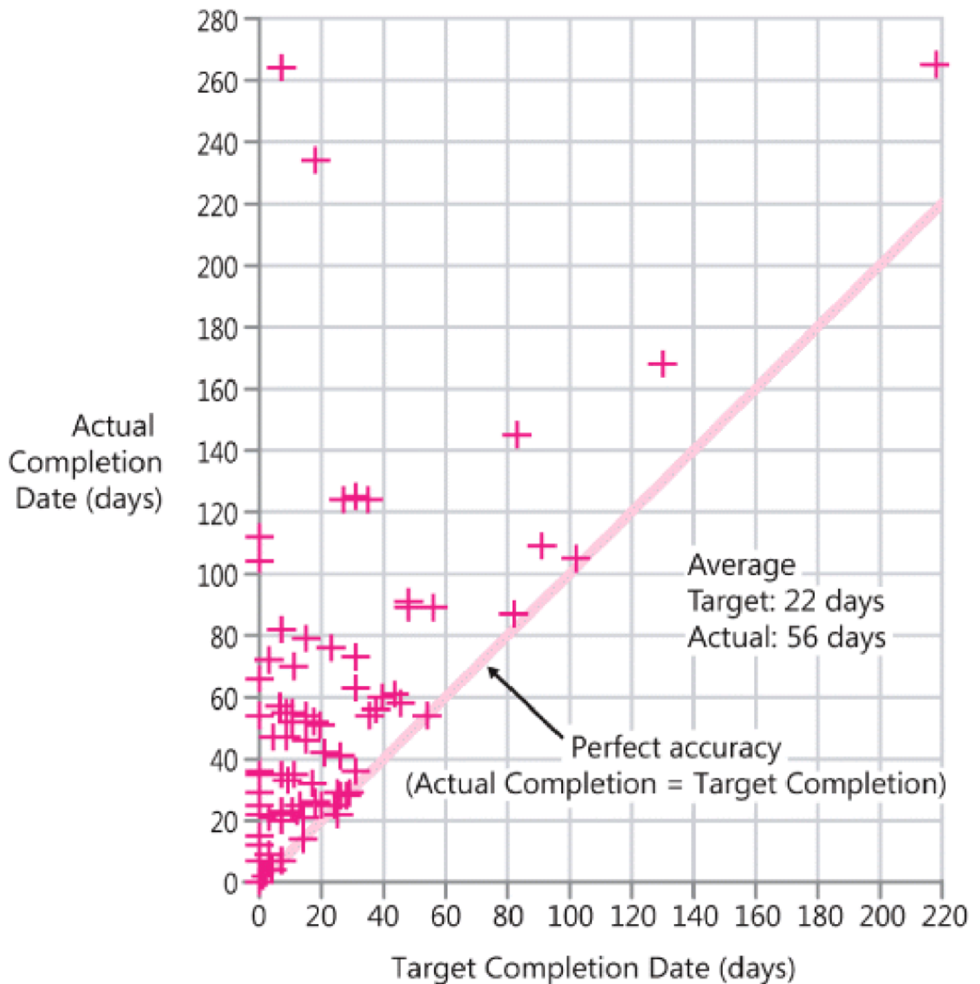


Figure 3-3. Estimation results from one organization. General industry data suggests that this company's estimates being about 100% low is typical. Data used by permission.

Copyrighted material

The points that are clustered on the “0” line on the left side of the graph represent projects for which the developers reported that they were done but which were found not to be complete when the software teams began integrating their work with other groups.

The diagonal line represents perfect scheduling accuracy. Ideally, the graph would show data points clustering tightly around the diagonal line. Instead, nearly all of the 80 data points shown are above the line and represent project overruns. One point is below the line, and a handful of points are on the line. The line illustrates DeMarco’s common definition of an “estimate”—the earliest date by which you could possibly be finished.

The Software Industry’s Systemic Problem

We often speak of the software industry’s estimation problem as though it were a neutral estimation problem—that is, sometimes we overestimate, sometimes we underestimate, and we just can’t get our estimates right.

But the software does not have a neutral estimation problem. The industry data shows clearly that *the software industry has an underestimation problem.* Before we can make our estimates more accurate, we need to start making the estimates *bigger*. That is the key challenge for many organizations.

Benefits of Accurate Estimates

Once your estimates become accurate enough that you get past worrying about large estimation errors on either the high or low side, truly accurate estimates produce additional benefits.

Copyrighted material

Improved status visibility. One of the best ways to track progress is to compare planned progress with actual progress. If the planned progress is realistic (that is, based on accurate estimates), it's possible to track progress according to plan. If the planned progress is fantasy, a project typically begins to run without paying much attention to its plan and it soon becomes meaningless to compare actual progress with planned progress. Good estimates thus provide important support for project tracking.

Higher quality. Accurate estimates help avoid schedule-stress-related quality problems. About 40% of all software errors have been found to be caused by stress; those errors could have been avoided by scheduling appropriately and by placing less stress on the developers (Glass 1994). When schedule pressure is extreme, about four times as many defects are reported in the released software as are reported for software developed under less extreme pressure (Jones 1994). One reason is that teams implement quick-and-dirty versions of features that absolutely must be completed in time to release the software. Excessive schedule pressure has also been found to be the most significant cause of extremely costly error-prone modules (Jones 1997).

Projects that aim from the beginning to have the lowest number of defects usually also have the shortest schedules (Jones 2000). Projects that apply pressure to create unrealistic estimates and subsequently shortchange quality are rudely awakened when they discover that they have also shortchanged cost and schedule.

Better coordination with nonsoftware functions. Software projects usually need to coordinate with other business functions, including testing, document writing, marketing campaigns, sales staff training, financial projections, software support training, and so on. If the software schedule is not reliable, that can cause related functions to slip, which can cause the

Copyrighted material

entire project schedule to slip. Better software estimates allow for tighter coordination of the whole project, including both software and nonsoftware activities.

Better budgeting. Although it is almost too obvious to state, accurate estimates support accurate budgets. An organization that doesn't support accurate estimates undermines its ability to forecast the costs of its projects.

Increased credibility for the development team. One of the great ironies in software development is that after a project team creates an estimate, managers, marketers, and sales staff take the estimate and turn it into an optimistic business target—over the objections of the project team. The developers then overrun the optimistic business target, at which point, managers, marketers, and sales staff blame the developers for being poor estimators! A project team that holds its ground and insists on an accurate estimate will improve its credibility within its organization.

Early risk information. One of the most common wasted opportunities in software development is the failure to correctly interpret the meaning of an initial mismatch between project goals and project estimates. Consider what happens when the business sponsor says, "This project needs to be done in 4 months because we have a major trade show coming up," and the project team says, "Our best estimate is that this project will take 6 months." The most typical interaction is for the business sponsor and the project leadership to negotiate the *estimate*, and for the project team eventually to be pressured into committing to try to achieve the 4-month schedule.

Bzzzzzt! Wrong answer! The detection of a mismatch between the project goal and the project estimate should be interpreted as incredibly useful, incredibly rare, early-in-the-project risk information. The mismatch indicates a substantial chance that the project will fail to meet its business

Copyrighted material

objective. Detected early, numerous corrective actions are available, and many of them are high leverage. You might redefine the scope of the project, you might increase staff, you might transfer your best staff onto the project, or you might stagger the delivery of different functionality. You might even decide the project is not worth doing after all.

But if this mismatch is allowed to persist, the options that will be available for corrective action will be far fewer and will be much lower leverage. The options will generally consist of "overrun the schedule and budget" or "cut painful amounts of functionality."

TIP #9

Recognize a mismatch between a project's business target and a project's estimate for what it is: valuable risk information that the project might not be successful. Take corrective action early, when it can do some good.

Value of Predictability Compared with Other Desirable Project Attributes

Software organizations and individual software projects try to achieve numerous objectives for their projects. Here are some of the goals they strive for:

- **Schedule**. Shortest possible schedule for the desired functionality at the desired quality level

Copyrighted material

- **Cost**. Minimum cost to deliver the desired functionality in the desired time
- **Functionality**. Maximum feature richness for the time and money available

Projects will prioritize these generic goals as well as more specific goals differently. Agile development tends to focus on the goals of flexibility, repeatability, robustness, sustainability, and visibility (Cockburn 2001, McConnell 2002). The SEI's CMM tends to focus on the goals of efficiency, improvability, predictability, repeatability, and visibility.

In my discussions with executives, I've frequently asked, "What is more important to you: the ability to change your mind about features, or the ability to know cost, schedule, and functionality in advance?" At least 8 times out of 10, executives respond "The ability to know cost, schedule, and functionality in advance"—in other words, *predictability*. Other software experts have made the same observation (Moseman 2002, Putnam and Myers 2003).

I often follow up by saying, "Suppose I could offer you project results similar to either Option #1 or Option #2 in Figure 3-4. Let's suppose Option #1 means that I can deliver a project with an expected duration of 4 months, but it might be 1 month early and it might be as many as 4 months late. Let's suppose Option #2 means that I can deliver a project with an expected duration of 5 months (rather than 4), and I can guarantee that it will be completed within a week of that date. Which would you prefer?"

Copyrighted material

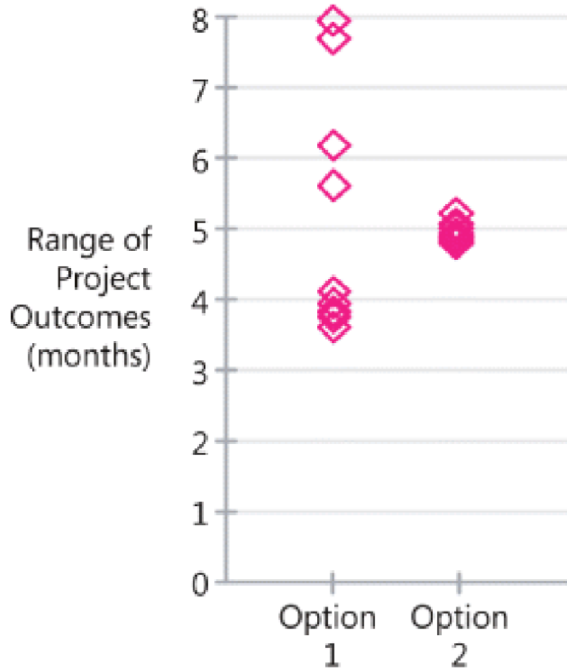


Figure 3-4. When given the option of a shorter average schedule with higher variability or a longer average schedule with lower variability, most businesses will choose the second option.

In my experience, nearly all executives will choose Option #2. The shorter schedule offered by Option #1 won't do the business any good because the business can't depend on it. Because the overrun could easily be as large as 4 months, the business has to plan on an 8-month schedule rather than a 4-month schedule. Or it delays making any plans at all until the software is actually ready. In comparison, the guaranteed 5-month schedule of Option #2 looks much better.

Copyrighted material

Over the years, the software industry has focused on time to market, cost, and flexibility. Each of these goals is desirable, but what top executives usually value most is predictability. Businesses need to make commitments to customers, investors, suppliers, the marketplace, and other stakeholders. These commitments are all supported by predictability.

None of this proves that predictability is the top priority for your business, but it suggests that you shouldn't make assumptions about your business's priorities.

TIP #10

Many businesses value predictability more than development time, cost, or flexibility. Be sure you understand what your business values the most.

Problems with Common Estimation Techniques

Considering the widespread poor results from software estimation, it shouldn't be a surprise that the techniques used to produce the estimates are not effective. These techniques should be carefully examined and thrown out!

Albert Lederer and Jayesh Prasad found that the most commonly used estimation technique was comparing a new project with a similar past project, based solely on personal memory. This technique was not found to be correlated with accurate estimates. The common techniques of "intuition" and "guessing" were found to be correlated with cost and

Copyrighted material

schedule overruns (Lederer and Prasad 1992). Numerous other researchers have found that guessing, intuition, unstructured expert judgment, use of informal analogies, and similar techniques are the dominant strategies used for about 60 to 85% of all estimates (Hihn and Habib-Agahi 1991, Heemstra and Kusters 1991, Paynter 1996, Jørgensen 2002, Kitchenham et al. 2002).

Chapter 5, presents a more detailed examination of sources of estimation error, and the rest of this book provides alternatives to these common techniques.

Additional Resources

[biblio03_001] Goldratt,EliyahuM. *Critical Chain*. Great Barrington, MA: The North River Press, 1997. Goldratt describes an approach to dealing with Student Syndrome as well as an approach to buffer management that addresses Parkinson's Law.

[biblio03_002] Putnam,LawrenceH. and WareMyers. *Five Core Metrics*. New York, NY: Dorset House, 2003. Chapter 4 contains an extended discussion of the importance of predictability compared to other project objectives.

Copyrighted material

Chapter 4. Where Does Estimation Error Come From?

> *There's no point in being exact about something if you don't even know what you're talking about.*
>
> *—John von Neumann*

A University of Washington Computer Science Department project was in serious estimation trouble. The project was months late and $20.5 million over budget. The causes ranged from design problems and miscommunications to last-minute changes and numerous errors. The university argued that the plans for the project weren't adequate. But this wasn't an ordinary software project. In fact, it wasn't a software project at all; it was the creation of the university's new Computer Science and Engineering Building (Sanchez 1998).

Software estimation presents challenges because estimation itself presents challenges. The Seattle Mariners' new baseball stadium was estimated in 1995 to cost $250 million. It was finally completed in 1999 at a cost of $517 million—an estimation error of more than 100% (Withers 1999). The most massive cost overrun in recent times was probably Boston's Big Dig highway construction project. Originally estimated to cost $2.6 billion, costs eventually totaled about $15 billion—an estimation error of more than 400% (Associated Press 2003).

Of course, the software world has its own dramatic estimation problems.

Copyrighted material

The Irish Personnel, Payroll and Related Systems (PPARS) system was cancelled after it overran its €8.8 million system by €140 million (The Irish Times 2005). The FBI's Virtual Case File (VCF) project was shelved in March 2005 after costing $170 million and delivering only one-tenth of its planned capability (Arnone 2005). The software contractor for VCF complained that the FBI went through 5 different CIOs and 10 different project managers, not to mention 36 contract changes (Knorr 2005). Background chaos like that is not unusual in projects that have experienced estimation problems.

A chapter on sources of estimation error might just as well be titled "Classic Mistakes in Software Estimation." Merely avoiding the problems identified in this chapter will get you halfway to creating accurate estimates.

Estimation error creeps into estimates from four generic sources:

- Inaccurate information about the project being estimated
- Inaccurate information about the capabilities of the organization that will perform the project
- Too much chaos in the project to support accurate estimation (that is, trying to estimate a moving target)
- Inaccuracies arising from the estimation process itself

This chapter describes each source of estimation error in detail.

Sources of Estimation Uncertainty

Copyrighted material

How much does a new house cost? It depends on the house. How much does a Web site cost? It depends on the Web site. Until each specific feature is understood in detail, it's impossible to estimate the cost of a software project accurately. It isn't possible to estimate the amount of work required to build something when that "something" has not been defined.

Software development is a process of gradual refinement. You start with a general product concept (the vision of the software you intend to build), and you refine that concept based on the product and project goals. Sometimes your goal is to estimate the budget and schedule needed to deliver a specific amount of functionality. Other times your goal is to estimate how much functionality can be built in a predetermined amount of time under a fixed budget. Many projects navigate under a happy medium of some flexibility in budget, schedule, and features. In any of these cases the different ways the software could ultimately take shape will produce widely different combinations of cost, schedule, and feature set.

Suppose you're developing an order-entry system and you haven't yet pinned down the requirements for entering telephone numbers. Some of the uncertainties that could affect a software estimate from the requirements activity through release include the following:

- When telephone numbers are entered, will the customer want a Telephone Number Checker to check whether the numbers are valid?
- If the customer wants the Telephone Number Checker, will the customer want the cheap or expensive version of the Telephone Number Checker? (There are typically 2-hour, 2-day, and 2-week versions of any particular feature—for example, U.S.-only versus international phone numbers.)

Copyrighted material

- If you implement the cheap version of the Telephone Number Checker, will the customer later want the expensive version after all?

- Can you use an off-the-shelf Telephone Number Checker, or are there design constraints that require you to develop your own?

- How will the Telephone Number Checker be designed? (Typically there is at least a factor of 10 difference in design complexity among different designs for the same feature.)

- How long will it take to code the Telephone Number Checker? (There can be a factor of 10 difference—or more—in the time that different developers need to code the same feature.)

- Do the Telephone Number Checker and the Address Checker interact? How long will it take to integrate the Telephone Number Checker and the Address Checker?

- What will the quality level of the Telephone Number Checker be? (Depending on the care taken during implementation, there can be a factor of 10 difference in the number of defects contained in the original implementation.)

- How long will it take to debug and correct mistakes made in the implementation of the Telephone Number Checker? (Individual performance among different programmers with the same level of experience varies by at least a factor of 10 in debugging and correcting the same problems.)

As you can see just from this short list of uncertainties, potential

Copyrighted material

differences in how a single feature is specified, designed, and implemented can introduce cumulative differences of a hundredfold or more in implementation time for any given feature. When you combine these uncertainties across hundreds or thousands of features in a large feature set, you end up with significant uncertainty in the project itself.

The Cone of Uncertainty

Software development consists of making literally thousands of decisions about all the feature-related issues described in the previous section. Uncertainty in a software estimate results from uncertainty in how the decisions will be resolved. As you make a greater percentage of those decisions, you reduce the estimation uncertainty.

As a result of this process of resolving decisions, researchers have found that project estimates are subject to predictable amounts of uncertainty at various stages. The Cone of Uncertainty in Figure 4-1 shows how estimates become more accurate as a project progresses. (The following discussion initially describes a sequential development approach for ease of explanation. The end of this section will explain how to apply the concepts to iterative projects.)

Copyrighted material

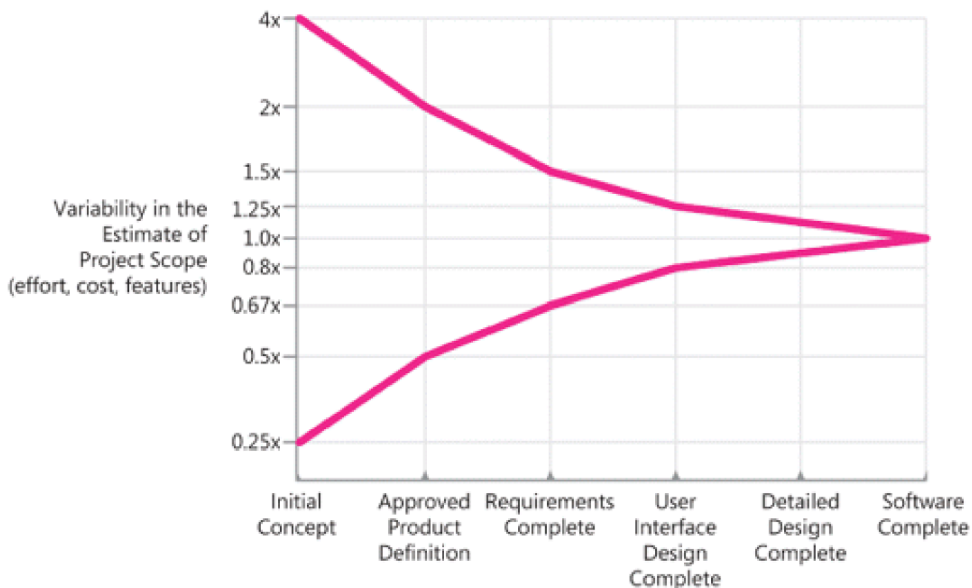


Figure 4-1. The Cone of Uncertainty based on common project milestones.

The horizontal axis contains common project milestones, such as Initial Concept, Approved Product Definition, Requirements Complete, and so on. Because of its origins, this terminology sounds somewhat product-oriented. “Product Definition” just refers to the agreed-upon vision for the software, or the *software concept*, and applies equally to Web services, internal business systems, and most other kinds of software projects.

The vertical axis contains the degree of error that has been found in estimates created by skilled estimators at various points in the project. The estimates could be for how much a particular feature set will cost and how much effort will be required to deliver that feature set, or it could be for

Copyrighted material

how many features can be delivered for a particular amount of effort or schedule. This book uses the generic term *scope* to refer to project size in effort, cost, features, or some combination thereof.

As you can see from the graph, estimates created very early in the project are subject to a high degree of error. Estimates created at Initial Concept time can be inaccurate by a factor of 4*x* on the high side or 4*x* on the low side (also expressed as 0.25*x*, which is just 1 divided by 4). The total range from high estimate to low estimate is 4*x* divided by 0.25*x*, or 16*x*!

One question that managers and customers ask is, “If I give you another week to work on your estimate, can you refine it so that it contains less uncertainty?” That’s a reasonable request, but unfortunately it’s not possible to deliver on that request. Research by Luiz Laranjeira suggests that the accuracy of the software estimate depends on the level of refinement of the software’s definition (Laranjeira 1990). The more refined the definition, the more accurate the estimate. The reason the estimate contains variability is that the software project itself contains variability. The only way to reduce the variability in the estimate is to reduce the variability in the project.

One misleading implication of this common depiction of the Cone of Uncertainty is that it looks like the Cone takes forever to narrow—as if you can’t have very good estimation accuracy until you’re nearly done with the project. Fortunately, that impression is created because the milestones on the horizontal axis are equally spaced, and we naturally assume that the horizontal axis is calendar time.

In reality, the milestones listed tend to be front-loaded in the project’s schedule. When the Cone is redrawn on a calendar-time basis, it looks like Figure 4-2.

Copyrighted material

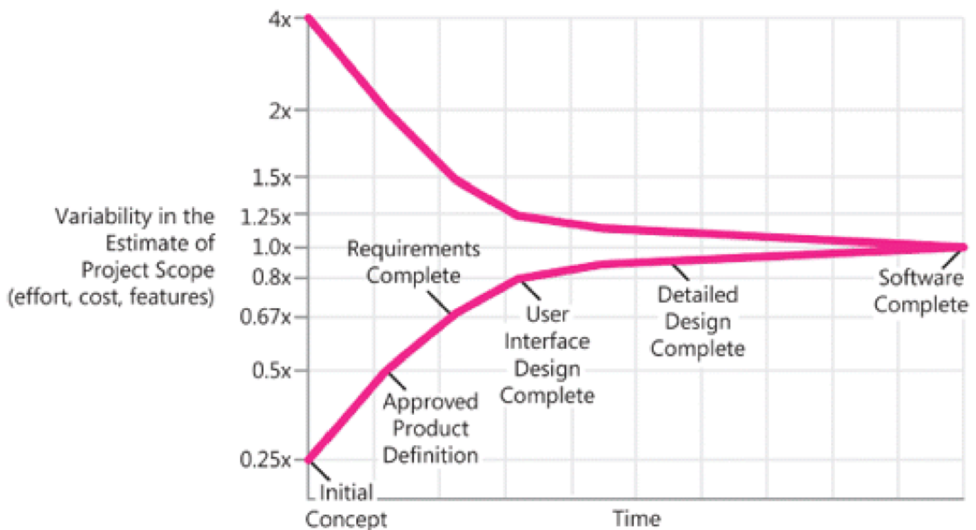


Figure 4-2. The Cone of Uncertainty based on calendar time. The Cone narrows much more quickly than would appear from the previous depiction in Figure 4-1.

As you can see from this version of the Cone, estimation accuracy improves rapidly for the first 30% of the project, improving from ±4*x* to ±1.25*x*.

Can You Beat the Cone?

An important—and difficult—concept is that the Cone of Uncertainty represents the *best-case accuracy* that is possible to have in software estimates at different points in a project. The Cone represents the error in estimates created by skilled estimators. It's easily possible to do worse. It isn't possible to be more accurate; it's only possible to be more lucky.

Copyrighted material

TIP #11

Consider the effect of the Cone of Uncertainty on the accuracy of your estimate. Your estimate cannot have more accuracy than is possible at your project's current position within the Cone.

The Cone Doesn't Narrow Itself

Another way in which the Cone of Uncertainty represents a best-case estimate is that if the project is not well controlled, or if the estimators aren't very skilled, estimates can fail to improve. Figure 4-3 shows what happens when the project doesn't focus on reducing variability—the uncertainty isn't a Cone, but rather a Cloud that persists to the end of the project. The issue isn't really that the estimates don't converge; the issue is that the project itself doesn't converge—that is, it doesn't drive out enough variability to support more accurate estimates.

Copyrighted material

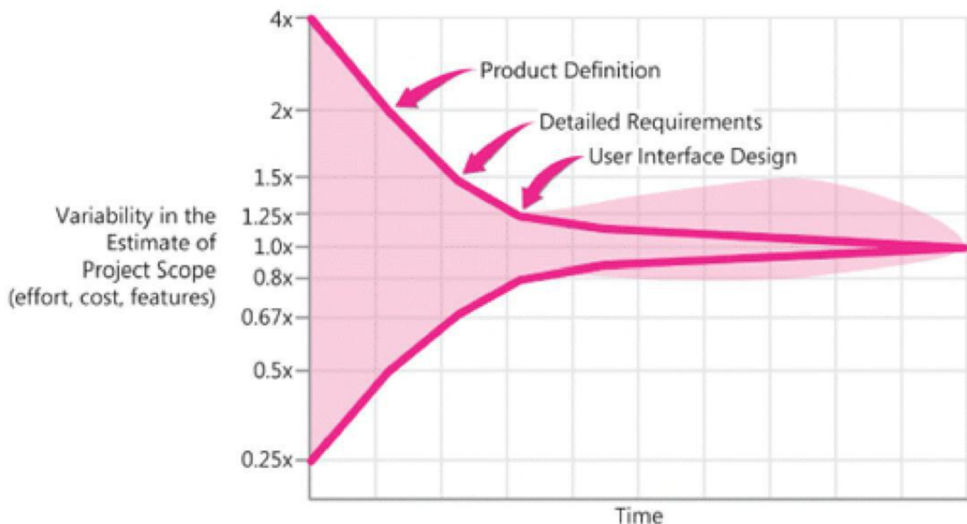


Figure 4-4. The Cone of Uncertainty doesn't narrow itself. You narrow the Cone by making decisions that remove sources of variability from the project. Some of these decisions are about what the project will deliver; some are about what the project will not deliver. If these decisions change later, the Cone will widen.

TIP #12

Don't assume that the Cone of Uncertainty will narrow itself. You must force the Cone to narrow by removing sources of variability from your project.

Accounting for the Cone of Uncertainty in Software Estimates

Copyrighted material

create the estimate you won't know whether the actual project outcome will fall toward the high end or the low end of your range.

TIP #13

Account for the Cone of Uncertainty by using predefined uncertainty ranges in your estimates.

A second approach is based on the finding that estimation "know-how-much" and estimation "know-how-uncertain" are two different skills. You can have one person estimate the best-case and worst-case ends of the range and a second person estimate the likelihood that the actual result will fall within that range (Jørgensen 2002).

TIP #14

Account for the Cone of Uncertainty by having one person create the "how much" part of the estimate and a different person create the "how uncertain" part of the estimate.

Relationship Between the Cone of Uncertainty and Commitment

Software organizations routinely sabotage their own projects by making commitments too early in the Cone of Uncertainty. If you commit at Initial Concept or Product Definition time, you will have a factor of 2*x* to 4*x* error

Copyrighted material

in your estimates. As discussed in Chapter 1, a skilled project manager can navigate a project to completion if the estimate is within about 20% of the project reality. But no manager can navigate a project to a successful conclusion when the estimates are off by several hundred percent.

Meaningful commitments are not possible in the early, wide part of the Cone. Effective organizations delay their commitments until they have done the work to force the Cone to narrow. Meaningful commitments in the early-middle part of the project (about 30% of the way in) are possible and appropriate.

The Cone of Uncertainty and Iterative Development

Applying the Cone of Uncertainty to iterative projects is somewhat more involved than applying it to sequential projects is.

If you're working on a project that does a full development cycle each iteration—that is, from requirements definition through release—you'll go through a miniature Cone on each iteration. Before you do the requirements work for the iteration, you'll be at the Approved Product Definition point in the Cone, subject to 4*x* variability from high to low estimates. With short iterations (less than a month), you can move from Approved Product Definition to Requirements Complete and User Interface Design Complete in a few days, reducing your variability from 4*x* to 1.6*x*. If your schedule is immovable, the 1.6*x* variability will apply to the specific features you can deliver in the time available, rather than to the effort or schedule. There are estimation advantages that flow from short iterations, which are discussed in Using Project Data to Refine Your Estimates.

What you give up with approaches that leave requirements undefined until the beginning of each iteration is long-range predictability about the

Copyrighted material

Common examples of project chaos include the following:

- Requirements that weren’t investigated very well in the first place
- Lack of end-user involvement in requirements validation
- Poor designs that lead to numerous errors in the code
- Poor coding practices that give rise to extensive bug fixing
- Inexperienced personnel
- Incomplete or unskilled project planning
- Prima donna team members
- Abandoning planning under pressure
- Developer gold-plating
- Lack of automated source code control

This is just a partial list of possible sources of chaos. For a more complete discussion, see Chapter 3, of my book *Rapid Development* (McConnell 1996) and on the Web at *www.stevemcconnell.com/rdenum.htm*.

These sources of chaos share two commonalities. The first is that each introduces variability that makes accurate estimation difficult. The second is that the best way to address each of these issues is not through estimation, but through better project control.

Copyrighted material

TIP #15

Don’t expect better estimation practices alone to provide more accurate estimates for chaotic projects. You can’t accurately estimate an out-of-control process. As a first step, fixing the chaos is more important than improving the estimates.

Unstable Requirements

Requirements changes have often been reported as a common source of estimation problems (Lederer and Prasad 1992, Jones 1994, Stutzke 2005). In addition to all the general challenges that unstable requirements create, they present two specific estimation challenges.

The first challenge is that unstable requirements represent one specific flavor of project chaos. If requirements cannot be stabilized, the Cone of Uncertainty can’t be narrowed, and estimation variability will remain high through the end of the project.

The second challenge is that requirements changes are often not tracked and the project is often not reestimated when it should be. In a well-run project, an initial set of requirements will be baselined, and cost and schedule will be estimated from that baselined set of requirements. As new requirements are added or old requirements are revised, cost and schedule estimates will be modified to reflect those changes. In practice, project managers often neglect to update their cost and schedule assumptions as their requirements change. The irony in these cases is that the estimate for the original functionality might have been accurate, but after dozens of new requirements have been piled onto the project—requirements that have

Copyrighted material

been agreed to but not accounted for—the project won't have any chance of meeting its original estimates, and the project will be perceived as being late, even though everyone agreed that the feature additions were good ideas.

The estimation techniques described in this book will certainly help you estimate *better* when you have high requirements volatility, but better estimation alone cannot address problems arising from requirements instability. The more powerful responses are project control responses rather than estimation responses. If your environment doesn't allow you to stabilize requirements, consider alternative development approaches that are designed to work in high-volatility environments, such as short iterations, Scrum, Extreme Programming, DSDM (Dynamic Systems Development Method), time box development, and so on.

> **TIP #16**
>
> To deal with unstable requirements, consider project control strategies instead of or in addition to estimation strategies.

Estimating Requirements Growth

If you do want to estimate the effect of unstable requirements, you might consider simply incorporating an allowance for requirements growth, requirements changes, or both into your estimates. Figure 4-5 shows a revised Cone of Uncertainty that accounts for approximately 50% growth in requirements over the course of a project. (This particular Cone is for purposes of illustration only. The specific data points are not supported by

Copyrighted material

the same research as the original Cone.)

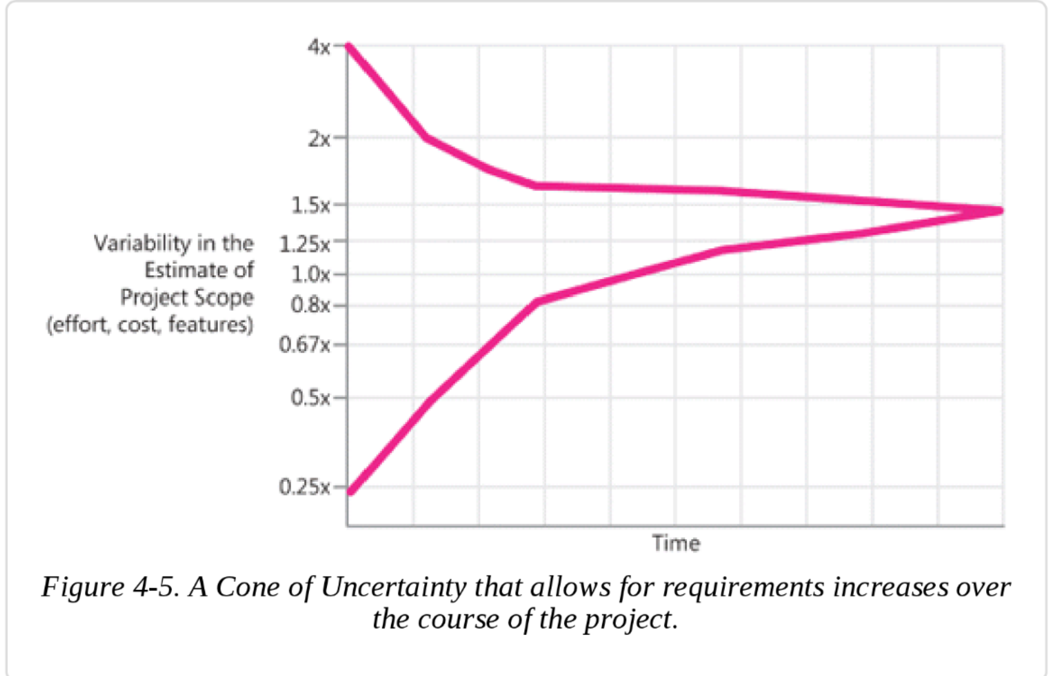


Figure 4-5. A Cone of Uncertainty that allows for requirements increases over the course of the project.

This approach has been used by leading organizations, including NASA's Software Engineering Laboratory, which plans on a 40% increase in requirements (NASA SEL 1990). A similar concept is incorporated into the Cocomo II estimation model, which includes the notion of requirements "breakage" (Boehm et al. 2000).

Omitted Activities

Copyrighted material

Table 4-3 lists software activities that estimators often overlook.

Table 4-3. Software-Development Activities Commonly Missing from Software Estimates

- Ramp-up time for new team members
- Mentoring of new team members
- Management coordination/manager meetings
- Cutover/deployment
- Data conversion
- Installation
- Customization
- Requirements clarifications
- Maintaining the revision control system
- Supporting the build
- Maintaining the scripts required to run the daily build
- Maintaining the automated smoke test used in conjunction with the daily build
- Installation of test builds at user location(s)
- Creation of test data
- Management of beta test program
- Participation in technical reviews
- Technical support of existing systems during the project
- Maintenance work on previous systems during the project
- Defect-correction work
- Performance tuning
- Learning new development tools
- Administrative work related to defect tracking
- Coordination with test (for developers)
- Coordination with developers (for test)
- Answering questions from quality assurance
- Input to user documentation and review of user documentation
- Review of technical documentation
- Demonstrating software to customers or users
- Demonstrating software at trade shows
- Demonstrating the software or prototypes of the software to upper management, clients, and end users
- Interacting with clients or end users; supporting beta installations at client locations
- Reviewing plans, estimates, architecture, detailed designs, stage plans, code, test cases,

Copyrighted material

what-if analysis, Things You Can Do with Tools That You Can't Do Manually

project requirements

creating, effort for, Estimating Allocation of Effort to Different Technical Activities, Estimating Schedule for Different Activities

Requirements Complete phase, Sources of Estimation Uncertainty, Accounting for the Cone of Uncertainty in Software Estimates

(see also)

iterative development, Accounting for the Cone of Uncertainty in Software Estimates

unstable (creeping), Chaotic Development Processes, Estimating Risk and Contingency Buffers

requirements omitted from estimates, Estimating Requirements Growth, Checklists

software to account for, Things You Can Do with Tools That You Can't Do Manually

project size, Details on the Software Industry's Estimation Track Record, Estimate Influences

Copyrighted material