IT Project Estimation

A Practical Guide to the Costing of Software

Paul Coombs
London

Cambridge University Press

PUBLISHED BY THE PRESS SYNDICATE OF THE UNIVERSITY OF CAMBRIDGE
The Pitt Building, Trumpington Street, Cambridge, United Kingdom

CAMBRIDGE UNIVERSITY PRESS
The Edinburgh Building, Cambridge CB2 2RU, UK
40 West 20th Street, New York, NY 10011-4211, USA
477 Williamstown Road, Port Melbourne, VIC 3207, Australia
Ruiz de Alarcón 13, 28014 Madrid, Spain
Dock House, The Waterfront, Cape Town 8001, South Africa

http://www.cambridge.org

First published 2003

Printed in the United States of America

Typefaces 10.75/13.5 pt. Berkeley Oldstyle and Franklin Gothic *System* LaTeX 2$_\varepsilon$ [TB]

A catalog record for this book is available from the British Library.

Library of Congress Cataloging in Publication Data
Coombs, Paul.
IT project estimation : a practical guide to the costing of software / Paul Coombs.
p. cm.
Includes bibliographical references and index.
ISBN 0-521-53285-X (pbk.)
1. Computer software – Costs. I. Title.
QA.76.76.C73 C66 2003
005.3 – dc21 2002191142

ISBN 0 521 53285 x paperback

Contents

Chapter 1

Introduction

> For which of you, intending to build a tower, sitteth not down first, and counteth the cost, whether he have sufficient to finish it? Lest haply, after he hath laid the foundation, and is not able to finish it, all that behold it begin to mock him, saying, This man began to build, and was not able to finish.
>
> —*Luke 14:28–30*

Are You Estimid?

No one wants to do the estimates. It is the most thankless task our industry can impose—an enormous responsibility for a difficult and highly speculative job. An unsuccessful estimate can result in long hours for the project team; sticky explanations to managers and customers; and, above all, enormous financial loss. If a project loses a million dollars, it might take ten million dollars worth of successful work to regain the break-even point. Everybody remembers the name of the person who costed a disaster, but no one ever recalls the genius whose prediction was correct to the day, for success will be attributed to the abilities and dedication of the development team. If you want glory: don't do the estimates.

So there is an understandable reluctance for anybody ever to provide any estimates. Most people will vacillate; delegate; or even, as a last resort, flatly refuse to provide a credible set of figures with their name appended. I call this

attitude **estimidity**. In this book, I discuss how estimidity is revealed and how it can be overcome. My aim is not to define some algorithmic method to obtain reliable estimates—indeed, it is my belief that no such method exists or will ever exist. Instead, my goal is to make you feel good about your own estimates by ensuring that they are the best that could be done at the time.

Bad estimates mean that good projects don't start, but land us with the impossible ones. So banish estimidity and tackle the task with thought and method to the best of your ability. No one can ask for more than that.

Why Are Estimates So Bad?

When I told some friends—all well-seasoned project managers—that I was writing this book, I was treated with some scepticism. "Things take as long as they take" was the general view. "All IT projects go over budget—it's just their nature." My friends all have plans, timetables, risk analyses, change control procedures, and the other manifestations of a well-managed project in place, but are cheerfully resigned to exceeding its estimated time and budget and to limiting the amount of the proposed functionality to be implemented.

They are not alone. The 2001 British Computer Society (BCS) Review revealed that of 1,027 projects surveyed, only 130 were successful—success being defined as delivering everything specified, to the quality agreed on, and within the time and costs laid out at the start. Of 500 development projects (rather than maintenance or data conversion), only 3 succeeded. In the United Kingdom, many recent public-sector IT ventures have ended as high-profile fiascos, but the story is similar in the private sector, in the United States, and in the rest of the world (see Chapter 12, "References and Resources", for more surveys).

Is all this down to poor estimation? We can divide the failures into two classes. Firstly, there are those where the rot had set in before the project even began—usually because it never was the right thing to do in the first place. Although this sad truth may emerge only midway through the development, we can't pin the failure onto bad estimation. But the second group are those projects which *could* have succeeded, but where the initial estimates disregarded the foreseeable risks. Most of the failure factors cited by project managers in the BCS survey are risks that they neglected to budget for—uncontrolled changes, unrealistic client expectations, open-ended third-party contracts, unexpected data conversions, complex interfaces, and so on. If risk is not eliminated in advance, it must be included in the budget. Instead, every time, the failure factors were encountered as a total surprise, and the project was derailed.

Why do we keep on making the same mistakes? I believe it is because underestimation is now expected and acceptable. If we costed projects properly, for example by adding sufficient contingency to cover all the risks, pricing staff at their real cost to the organisation, and including the running costs of the completed system, then most would never start; they could not be justified. But everyone *wants* new projects to start. Managers have revolutionary ideas; there are business opportunities that can't be missed; and, let's face it, designing new systems is far more interesting than maintaining the old. And as for asking some consultants, well, of course they think you should proceed—indeed, they will often quote you a price below their cost, secure in the knowledge that there will be sufficient changes to recoup the initial loss. The friends I mentioned earlier started their projects *knowing* that success was unlikely, and possibly not even expected, so they are not striving overmuch to achieve it. They see their job as one of damage limitation.

Here's some proof. Several years ago, I undertook an analysis of the fixed-price projects undertaken by a large software house. The results are summarised in Figure 1.1. Some of the projects made large profits, and some made large losses, but most clustered around the break-even point. At first sight, this may seem fine—the majority of projects completed for somewhere near the cost estimated. But the *Y* scale is the profit, not the cost. The software house added a

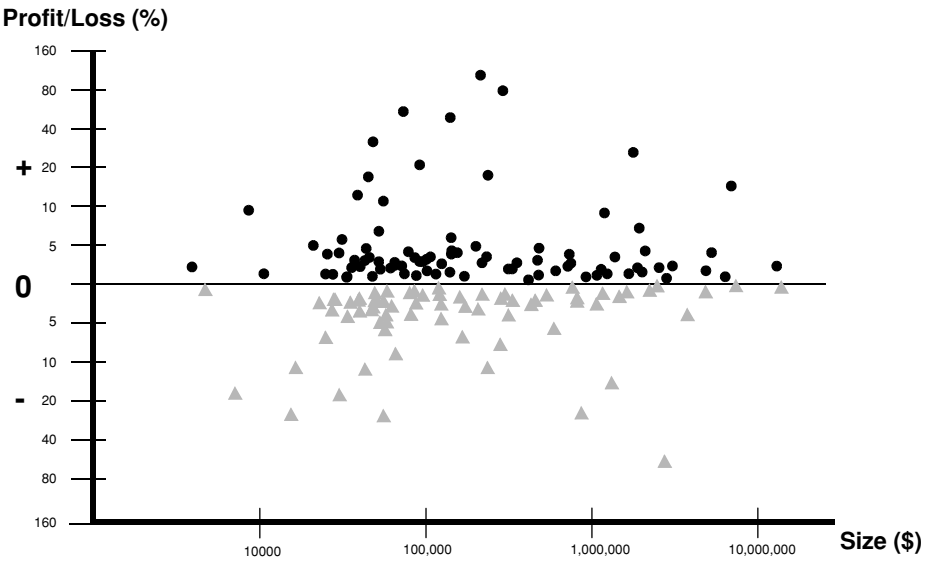

FIGURE 1.1. Analysis of Fixed-Price Projects

mark-up and a contingency allowance in order to derive the fixed price quoted to the customer. So the chart shows that most of these projects ate all the time estimated, ate all their contingency allowance, and finally ate all the potential profit before stopping. What has applied is a corollary to Parkinson's Law: The work has contracted to fit the budget available. How? By reducing the functionality, the quality, or both. As the bottom of the pot was reached, the more accurate the cuts became, so each project eventually delivered something just about acceptable for the price.

There are many conclusions we can draw from this chart. "Stop doing fixed-price projects" is one, for there seems to be no profit in them. And "don't bother with estimates" is another, for it appears that most projects will deliver *something*, regardless of the allocated budget. But what the chart really demonstrates are the points I have already made—most projects are underestimated to start with, insufficient allowance is made for predicable risks, and the only option left to project managers is to take an axe to the requirements. Underestimation may not explain every outcome, but if all the dots on the chart were raised upwards by a few notches, the software house would be undertaking work that yielded a consistent and predictable profit, and their customers would receive the functionality and quality they specified. Accurate estimates get us what we all want, but over-optimistic figures dreamed up to make a sale, to force through a pet idea, or to meet artificial delivery targets eventually lead to compromise.

Many years ago, I watched as a nuclear power station was constructed in one of my favourite parts of the countryside. The project was meticulously planned. The team did not dig a huge hole in the ground and then say, "Hmm, what shall we do now? Nuclear, gas, or coal-fired? Pressurised water or gas-cooled?" No—only *software* engineers work like that. "Let's build the easiest, cheapest, flashy, interesting, beneficial, cost-effective, or quickest bit first" has come the cry from a thousand different projects, "and we'll worry about the rest later." I have lost count of the times I have costed a system "properly" only to have that figure butchered in order to justify commencement. How often have we started on "phase one of phase one"—a project yielding no business benefit in itself, but one that allows us to make a start on a larger initiative for a low cost? And how many times have we then found that because that project yielded no benefit, obtaining the funding for the next phase became impossible, and the initiative flopped?

We need to get real. The purpose of an estimate is not to come up with a price that will get the project off the ground. So let's examine why we really need to undertake this thankless task.

Why Estimate?

Estimates are needed for three principal reasons:

1. To justify a project—enabling the cost to be compared with the anticipated benefits, and "what if" assessments to be undertaken in order to choose between different technical, environmental, or functional options.
2. To act as the central element of software engineering practice—enforcing the discipline to make the project succeed.
3. To improve the software production procedures—evaluating the effects of process improvements.

I expand on each of these in turn.

Justifying the Project

I once undertook some work for a large organisation that initiated new business process reengineering programmes after each management shake-up. No justification was attempted—the organisation did not maintain the underlying figures to allow costs or benefits to be assessed. Instead, the gut feel of the new architect of change was sufficient. The initiative would potter along for a year or two, and then the sheer volume of work, and the associated costs, would suddenly become apparent. The programme would be halted and then renamed, and soon there would be another management shake-up, allowing the cycle to begin again. That organisation is now in crisis because it has not modernised its business model, despite the expenditure of countless millions on projects intended to do so. A familiar story, with a simple moral—*estimation saves money*. In this case, it would have been difficult and expensive (the first time) to analyse the costs and benefits of any proposed initiative, but well worthwhile in order to save the money wasted on work that could not be justified.

For projects can only be justified if they are cost-effective and timely. If a new system saves one million dollars a year but costs forty million dollars to develop, it's probably not going to be approved. If the software has to display the election results or control a satellite, it better be finished on time or the entire effort will be wasted. I can think of very few cases in history where a major engineering development took place with no regard to cost or time. Even while the Great Pyramid was being planned I'm sure that someone wanted to know if the expense was justified and that they continued to question this while the

effort mounted and the timescales lengthened. The business, organisational, and technical environment will inevitably change as the project proceeds, so there may be a point at which continuation is no longer worthwhile. By undertaking an honest, unbiased estimate we can see if it is sensible even to begin, and if we keep this view of the costs and benefits up to date, we can see if it is worthwhile to carry on.

This book concentrates on costing. If you want to decide if your project should or should not go ahead, or whether a change is justified, you also need to evaluate the benefits. And you must be able to do that using a measurement that is directly comparable with the cost: money. It may seem difficult to estimate the cost of a software application, but estimating the benefits it brings in cash terms is often even harder. How many more people will buy our product if it has this extra feature? How many will stop using it if the feature is not implemented? But there is no point in estimating the project cost to the nearest cent if this is going to be compared with someone's gut feel of the benefits. Improvements to the costing of proposed projects must go hand-in-hand with improvements in determining their benefits.

Engineering the Development

At the nuclear power station, the engineers did not complete the reactor and then decide to order some fuel rods, which take three years to make. They didn't complete the roof first and then start on the walls. The work of the different teams had to be synchronised, equipment and components had to arrive at the anticipated time, and operational staff had to be trained and ready—just as for most IT projects. If each of these parallel strands is not predicted and controlled, the timescale will start to stretch, and the costs to rise.

Software development is an engineering task. Sometimes highly skilled and motivated technical teams do manage to craft innovative systems with the minimum of process and planning, but the rest of us need a disciplined framework within which we can make progress. This means establishing processes for requirements management, change control, progress assessment, testing methodology, and all the other elements of the development life cycle. Even if incremental techniques are being used, there must be an engineering approach within each cycle, or we end up with technical anarchy.

So where do estimates fit into the engineering paradigm? Everywhere. You can't manage what you don't measure, and to measure something you must have a standard to measure against. So at every stage of the project engineering life

cycle, from the vague concept to the post-mortem, an up-to-date model of the anticipated time and cost is needed in order to direct the team and to inform the customers. Without estimates, you do not have such a model. Its purpose is not make the project finish any earlier but to allow everyone to see what has happened, what is happening, and—most importantly—what is yet to happen. We must start with an initial estimate—our model of the cost and timescale of the project—calibrating and maintaining this as development proceeds and the inevitable problems and changes are encountered.

For during software engineering projects, it is taken for granted that serious changes to the functionality can be introduced at any stage, while still maintaining the same timescale. Civil engineers would not alter a bridge design from suspension to cantilever while halfway across, but IT project managers are well used to changes just as fundamental. The British Computer Society review found that 76.3 percent of project managers reported that they had *never seen* an IT project delivered in accordance with its initial specifications. How do our designs get so out of control? Obviously, one reason is simply that you *can* alter software specifications more readily than bridge blueprints. And the environment into which IT systems are delivered changes more quickly than the equivalent for bridges. But the most significant reason is that we so rarely evaluate the cost of a proposed change or how much benefit it will bring. Instead, we depend on someone's gut feel that it is the right thing to do. It is *because* it is possible to change IT projects so easily that we need estimates. They give us some ammunition—some real facts—with which we can evaluate whether a particular development, or a proposed modification, is justified.

Improving the Process

Other engineering disciplines have managed to introduce predictability through the sharing of experience, standardisation, and modelling. For example, although some civil engineering projects (such as the Channel Tunnel) are unique, high risk, and prone to overruns, most (such as apartment blocks and bridges) are completed on time, within budget, and don't keep falling down once delivered. But software engineering projects all seem to be of the overrunning, falling-down-afterwards type. We should be getting better, utilising component-based and off-the-shelf solutions, but we still seem to come up against the same old difficulties, both technical (e.g., communications, integration, and testing) and organisational (e.g., customer expectations, change management, and third

parties). And we never learn. If an aircraft crashes or a bridge collapses, there is an investigation, a report, and the whole industry learns a lesson. Software engineering disasters are just written off, explained away, and then ignored.

We need to come out of denial. By measuring how long a project takes, comparing this against the original estimates, and analysing the differences, we will improve our techniques for estimation. And we can then see, in a tangible form, the results of using new tools and methods. Every organisation needs metrics in order to assess how well they are doing against expectations. In an organisation where IT projects play an important role, the metrics associated with those projects are particularly essential. So collect them. You will then have the tools to improve the predictability of IT initiatives, while your competitors will continue to cover up the disasters, ignore the lessons, and repeat the same mistakes.

Is Estimation Possible?

There are twelve **Blindingly Obvious Rules of Estimation** defined in this book. I have called them that because they are not rocket science, not precise results from years of academic research, and not even truths only someone as experienced as myself could have realised. They are rules we all know. Of the twelve, Rule 1 is not only the first but also the most fundamental.

Blindingly Obvious Rule of Estimation Number 1
Your estimate will be wrong.

How can it be otherwise? You are being asked to predict the future. And it is not the future of something with so limited a count of contributory factors and end results as a horse race. It is an IT project—an undertaking that may encounter any number of events that will affect its progress, change its scope, and challenge its entire reason for being. No one can foresee what will happen to such a beast.

Even if you implemented the same project several times over, it would take a different course each time. So, when you think about it, the project duration is not a fixed number but a statistical entity, as shown in Figure 1.2. There is a minimum time the project could possibly take, which is why the standard bell curve is skewed. But there are so many events that may happen along the way that we could never predict the maximum duration. So estimation—or at least

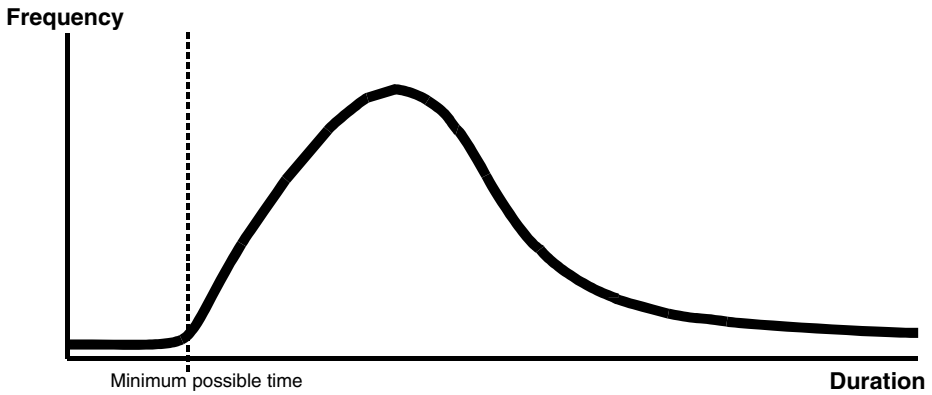

Figure 1.2. Frequency of Possible Project Durations

100 percent accurate estimation—is not possible. You could pick a number and be lucky, but in a parallel universe, fate would take a different course and that number would be wrong.

Despite this, we can use our best judgment, taking into account every fact and every doubt we have, to produce the most likely figure that can be predicted at the time. But don't ever convince your management, your customers, or yourself that this number is going to be precise to the day or to the dollar.

What's Wrong with Overestimation?

Estimidity encourages the belief that it is never wrong to overestimate. But this pretty much guarantees that new projects will never start or, in a commercial environment, that you will never win any new work. It may not be your decision to withhold the go-ahead, but if you have padded the estimates just to protect your back then you will bear responsibility for the failure to achieve the benefits that the project could have delivered.

In IT's early days, the management of large software projects was somewhat hit-and-miss because the drivers behind success or failure were unknown and unpredictable. Overestimation was common, because profitability could still be achieved even if there was time to draw a few "Snoopy" calendars along the way. Today's world is more competitive, and organisations can no longer afford to finance projects that will *easily* run to time and budget. So we need precision in our estimates, not just the figure for which we can say, "it can't possibly take longer than that."

Who Should Do the Estimates?

The best people to undertake the estimates are the ones who are going to undertake the work. Clearly, they must have some relevant experience or they are going to have no baseline with which to compare the new project. Conversely, there can be no role for a "professional estimator" because the technical experience of such a person would soon become so out-of-date as to be worthless. I undertake many estimates, but I have to match that task with substantive project work, or else by now I would be mapping mainframe/dumb terminal assumptions onto server/browser applications.

So the ideal estimator is the project manager or technical manager who has been nominated for the work in hand. He or she may draw on specific experience from the proposed team—the developers who will actually be doing the work—but someone is needed to collate the results and take responsibility for the total. If there are projects with which this one can be validly compared, some expertise in the application area, or any familiarity with the technology to be used, the relevant people need to be brought into the estimation team or made available for consultation.

Where the professional estimator *is* useful is in the promotion of consistent technique. If everyone in your organisation has their own secret way of deriving estimates, and these are never questioned or the outcome validated, you will never get any better at sizing new projects. The methodology must be open, consistent, subject to challenge at a review, and calibrated at the end of each project. Someone needs to make sure that this is the case; in larger organisations this is a full-time role for someone within the software engineering function.

When Can an Estimate Be Provided?

A few years ago, while I was working in a software house, I was passed in the corridor by a senior manager. "How long do you think it would take to do a C++ compiler?" he asked by way of greeting. I gave a number of person-years, and we passed without a further word. I don't know why he wanted this number, and probably never will. The point is that he probably didn't know if such a project would take one or one thousand years. He could have been on his way to a meeting where that lack of knowledge could have made both him and our company look so inept as to be unemployable. Many people would not have given that manager a straight answer to his query, and I can see why.

"What machine?", "Where?", "What version of C++?", and "What have we got to start with?" are just some of the questions I could have asked, and so wriggled out. But that was not an option; my manager wanted a number that minute. Not to have provided one would be estimidity in action. So we come to Rule 2.

Blindingly Obvious Rule of Estimation Number 2
You can always provide an estimate.

I don't care if there are hundreds of things about the project that you don't know (e.g., what it does, what the performance requirements are, what the platform is, what methodology you're using, what the language is, or what level of testing is needed), it is always possible to come up with some number, regardless of the number of caveats you may care to set with it.

Army privates have a saying that there is no greater danger than an officer with a map; I have that same feeling about managers and estimates. We all know that your number will be cast in stone, and the caveats will be forgotten. Such is life. You may argue that a figure plucked from the air has no validity—a guess is not an estimate. But my manager chose me to help with his problem because he knew that I had the experience to provide an estimate of sufficient accuracy for his purpose. To evade such a responsibility under a smokescreen of excuses would not help anyone.

Where Do You Start?

Techniques for estimation fall into four categories:

1. **Expert Judgment**—Consult with one or more experts, who use their experience to arrive at an estimate.
2. **Analogy**—Compare the proposed project with one or more completed systems, analysing similarities and differences in order to derive the estimate.
3. **Bottom-Up**—Decompose the work into its components, estimate each of these individually, and then sum the results to obtain an overall figure.
4. **Algorithmic**—Use a mathematical model to derive the cost or timescale. Input parameters define the unique characteristics of the project, and these are fed into a set of equations in order to obtain an estimate.

Expert judgment is fine, so long as you can find an expert whom you trust. According to Delia Smith's *Complete Cookery Course*, it should take me three hours to make a steak-and-kidney pie. But does this mean I should start three hours before my dinner party begins, or should I be building in factors unique to my own situation—such as a lack of equipment, patience, and cooking ability? So choose your expert well, and don't accept his or her opinion unless it is directly applicable to your own environment and methods of working. That said, I would guess that the reason you are reading this book is that you are supposed to *be* one of the experts. Maybe you can get some specialist help, but you will also need some other techniques.

If I have made a chicken casserole several times before, I could conclude that it would take around the same time to cook my steak-and-kidney pie. Estimation by analogy is so psychologically appealing that it is tempting to see all kinds of previous projects as having some degree of commonality with the one proposed. But as the new project develops, the similarities fade away, and the task manifests unique and troublesome characteristics of its own. For my pie, I find that beef takes longer to cook than chicken, and making some pastry turns out to be more difficult than it looks. We may say, "The XYZ system took sixty person-years, and this one is pretty much the same, but with different technology, a less experienced team, and a new application area—so call it seventy person-years." But this is not going to be very accurate. It may do for a first pass, but could you then go on to detail all the differences and estimate these individually in order to obtain more precision? This is not to say that previous experience should be ignored. The more precedents and analogies we can incorporate, the more accurate our estimate will be; later in this book I show how this knowledge can be most effectively leveraged. But such expertise is better applied at a lower level rather than to the estimate as a whole. Estimation by analogy is fine for order-of-magnitude assessments, but only in cases where the projects really are directly comparable.

That's all I want to say about the first two categories of estimation technique; the remainder of this book concentrates on the bottom-up and algorithmic methods. But the best approach is a combination of all four. Assemble the most experienced team you can, employ analogies to get a feel for the overall problem, use the bottom-up technique to increase the accuracy, and deploy algorithmic tools if you think they will help to confirm the result. Above all, don't try to economise by using unqualified people, allowing insufficient time, prejudging the result, or failing to build on your previous experience. The penalties of inaccurate estimation are so high that it is worth an investment to get the best result possible.

What Is Contingency?

The Dilemma

How long does it take you to get to work in the morning—from shutting your front door to settling down in your office chair? Very few of us could state a fixed time, say, forty-eight minutes. It *would* be forty-eight minutes if your car starts, if you don't forget your bus pass, if your train is on time, if you can find a space in the traffic to cross the road, and if you are not abducted by aliens from the planet Zog. So you could say it is "fifty-five minutes on average" or "usually somewhere between forty-eight and sixty-five minutes." In fact, there is no top limit; there are factors—**risks**—that could ensure that you never reached work at all.

Now suppose I asked you to give me an estimate of how long it will take you to get to work *next Monday*, and I will fine you a dollar for each minute you are out either way. Clearly, "forty-eight minutes" is not a good answer, for you have to take some account of the risks. On the other hand, you can't assume that *every* risk will occur, or you'll end up owing me a lot of money should you happen to arrive after forty-eight minutes. This is the dilemma of project estimation. Determining a minimum figure is hard enough, but we also need to make an allowance for the risks. If we make too little allowance then these hazards may be encountered and the project will be late, but if we add too much then it may not seem worthwhile to start the project at all.

So the existence of risk means that our estimate will always be uncertain. We cannot ignore this uncertainty, but must embrace it within the estimation and planning process. This is Rule 3.

Blindingly Obvious Rule of Estimation Number 3
Every estimate must have a contingency allowance.

An estimate consists of two figures, the **base** and the **contingency**. They are as inseparable as the x and y values in a pair of coordinates. The contingency is an estimate in itself, of the amount of trust that that you are placing in the base value. This is not the same as a degree of tolerance, the way a 100-Ω resistor may be ±1 percent. The resistor may be 99-Ω, but a task estimated at twenty days with 50 percent contingency is never going to take ten days. What that pair of figures says is that the task will take at least twenty days, and the best allowance to make is thirty days.

Separating Risk from the Base Value

Some people have more risky journeys to work than others. If you live nearby and walk in, your estimate for next Monday may be ten minutes plus two minutes' contingency. Or you may take several trains and be highly dependent on whether you make the connections—maybe forty-five minutes plus thirty minutes' contingency. Similarly, some projects are more risky than others. We may have undertaken a similar development many times before and therefore have a procedure for pretty much everything that could happen. Or we may be venturing into the unknown, not really understanding either the problem or the solution. The point is that the risk is reflected *only* in the contingency, not the base value. The latter reflects how long we think things will take if everything goes well, and the contingency is the allowance we will make for the untoward.

Separating the base value from the contingency actually makes the estimation process easier. It's still hard, mind you, but it does mean that we can determine our base value without needing to take account of all the bad things that may happen, and also that we can use some risk analysis techniques to determine the right level of contingency.

The Sweet Spot

That "right level" is dependent on the amount of risk you wish to take. Remember that the estimate is a statistical value—a point on the bell curve of possible

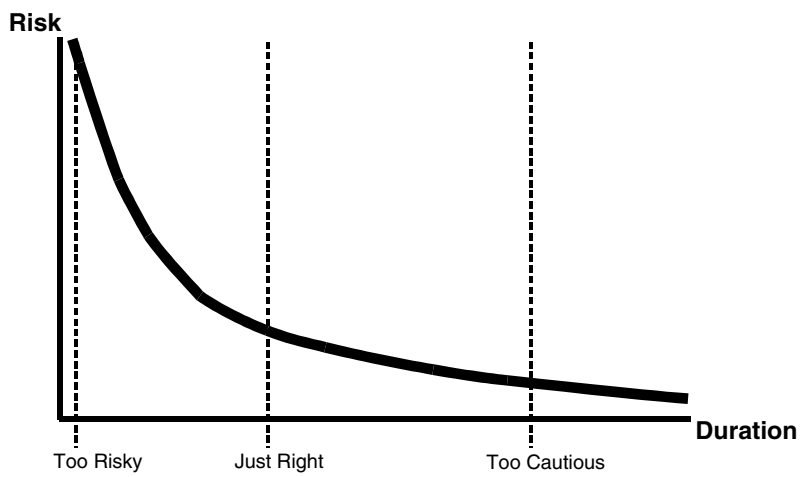

FIGURE 1.3. Estimates Incorporating Differing Levels of Risk