



# PROJECT AIR FORCE

THE ARTS

CHILD POLICY

CIVIL JUSTICE

EDUCATION

ENERGY AND ENVIRONMENT

HEALTH AND HEALTH CARE

INTERNATIONAL AFFAIRS

NATIONAL SECURITY

POPULATION AND AGING

PUBLIC SAFETY

SCIENCE AND TECHNOLOGY

SUBSTANCE ABUSE

TERRORISM AND  
HOMELAND SECURITY

TRANSPORTATION AND  
INFRASTRUCTURE

WORKFORCE AND WORKPLACE

This PDF document was made available from [www.rand.org](http://www.rand.org) as a public service of the RAND Corporation.

[Jump down to document](#) ▼

The RAND Corporation is a nonprofit research organization providing objective analysis and effective solutions that address the challenges facing the public and private sectors around the world.

## Support RAND

[Purchase this document](#)

[Browse Books & Publications](#)

[Make a charitable contribution](#)

## For More Information

Visit RAND at [www.rand.org](http://www.rand.org)

Explore [RAND Project AIR FORCE](#)

View [document details](#)

## Limited Electronic Distribution Rights

This document and trademark(s) contained herein are protected by law as indicated in a notice appearing later in this work. This electronic representation of RAND intellectual property is provided for non-commercial use only. Permission is required from RAND to reproduce, or reuse in another form, any of our research documents.

This product is part of the RAND Corporation monograph series. RAND monographs present major research findings that address the challenges facing the public and private sectors. All RAND monographs undergo rigorous peer review to ensure high standards for research quality and objectivity.

# Software Cost Estimation and Sizing Methods

Issues and Guidelines

---

Shari Lawrence Pfleeger, Felicia Wu, Rosalind Lewis

Prepared for the United States Air Force

Approved for public release; distribution unlimited



PROJECT AIR FORCE

The research described in this report was sponsored by the United States Air Force under Contract F49642-01-C-0003. Further information may be obtained from the Strategic Planning Division, Directorate of Plans, Hq USAF.

**Library of Congress Cataloging-in-Publication Data**

Pfleeger, Shari Lawrence.

Software cost estimation and sizing methods: issues and guidelines /  
Shari Lawrence Pfleeger, Felicia Wu, Rosalind Lewis.

p. cm.

“MG-269.”

Includes bibliographical references.

ISBN 0-8330-3713-7 (pbk. : alk. paper)

1. Computer software—Development—Management. I. Wu, Felicia. II.  
Lewis, Rosalind. III. Title.

QA76.758.P4797 2005

005.1—dc22

2004030691

The RAND Corporation is a nonprofit research organization providing objective analysis and effective solutions that address the challenges facing the public and private sectors around the world. RAND’s publications do not necessarily reflect the opinions of its research clients and sponsors.

**RAND**® is a registered trademark.

© Copyright 2005 RAND Corporation

All rights reserved. No part of this book may be reproduced in any form by any electronic or mechanical means (including photocopying, recording, or information storage and retrieval) without permission in writing from RAND.

Published 2005 by the RAND Corporation

1776 Main Street, P.O. Box 2138, Santa Monica, CA 90407-2138

1200 South Hayes Street, Arlington, VA 22202-5050

201 North Craig Street, Suite 202, Pittsburgh, PA 15213-1516

RAND URL: <http://www.rand.org/>

To order RAND documents or to obtain additional information, contact

Distribution Services: Telephone: (310) 451-7002;

Fax: (310) 451-6915; Email: [order@rand.org](mailto:order@rand.org)

# Executive Summary

---

## Introduction (see pp. 1–7)

Estimating the size and cost of software is a risky business. When software is a crucial component in numerous space, weapon, aircraft, and information technology projects critical to operations, as it often is for the Air Force, accurate estimates of software costs are essential. Because software size is usually the most influential factor in determining software costs, good estimates of size are critical to good cost estimation. Rather than seeking the perfect method for estimating size and cost exactly, a more realistic approach to improving estimation is to reduce the risks (that is, to anticipate likely problems) associated with improper sizing and costing of software.

Consequently, the goal of this report is to aid experienced cost analysts in understanding the sources of uncertainty and risk in sizing and costing software, and to provide insight into mitigating the risks when making choices about different sizing and costing options. We pay particular attention to the early stages of a project, when many of the factors needed to support estimation (such as the particulars of each system requirement) may be unknown or uncertain.

The notion of risk is central to any such analysis, and two techniques can improve accountability of risks relating to software estimates: identifying areas of uncertainty (that may lead to risky situations) and analyzing the estimation process to determine where

risk mitigation can reduce the uncertainty. The first technique increases an analyst's diligence in reporting uncertainty. The second technique involves actually addressing and mitigating risks in the estimation process, thereby reducing the total uncertainty and increasing the estimate's accuracy. The two techniques are complementary. The first improves accountability by reporting the uncertainty. The second improves accountability by dealing with and reducing the uncertainty.

This document addresses both techniques, offering guidelines to cost analysts on how best to manage the unavoidable risks that are attendant on predicting software size and cost. These techniques inject realism into the estimation process, acknowledging that estimates are often made with limited knowledge of the system and a profusion of choices that may be rife with uncertainty.

## **Sizing Methods (see pp. 9–13)**

Software size estimation is critical to providing a credible software cost estimate; thus, choosing the appropriate method by which to estimate size is important. In most cases, the estimation risk (that is, the possibility that the estimate will be far different from the actual software cost) depends more on accurate size estimates than on any other cost-related parameter. Thus, it is important that software sizing be done as consistently and accurately as possible, given the uncertainties inherent in estimation.

However, software sizing is difficult for a number of reasons. First, it is performed in a variety of different contexts,<sup>1</sup> some with a great deal of knowledge about the system and some with almost no knowledge at all. Second, there are many choices for the language and structure used to express the requirements and design. Third, software projects are often a combination of new, reused, and modified

---

<sup>1</sup> The context depends on the resources available to the project, the degree to which the developers are familiar with the problem to be solved by the software, the developers' expertise in the problem domain and with the development tools, and more.

components. A sizing method must be able to incorporate all three modes, even when the reuse and modification occur in the requirements and design instead of just in the code.

Both sizing and costing methods typically belong to one of two types, or a combination of the two types: expert judgment or measurable items. The expert judgment method relies on the ability of one or more analysts to determine the likely product size by evaluating the nature of the requirements, often in some qualitative fashion. Usually, the analysts have knowledge of similar development efforts, and the degree of similarity is relative to their understanding of the proposed project. By contrast, sizing based on quantitative, measurable items can use aspects of the requirements, such as number of requirements, number of transactions and screens, or other constructs (such as function points), to suggest the resulting size. With this approach, the size-estimation process is often more formal; the analysts are guided by questions or steps to elicit parameters from which the likely size is then calculated.

## **Advantages and Disadvantages of Sizing Methods**

Several global issues should be considered when using a sizing method. We discuss them in the following categories (see pp. 13–22):

- Counting physical objects, such as lines of code or number of requirements. Advantages include ease of counting (and ease of counting automation), independence of programming language, ease of storage in a historical database, and ease of management understanding. Disadvantages include difficulty of counting early in the development process, dependence on programming or specification style, need for rigor in applying counting rules, and inconsistency of methods across different languages.
- Counting notional constructs, such as function points or application points. These objects may be easier than physical objects to define early in the development process, but as notional ideas they are often more difficult to track over the course of devel-

opment. Advantages include ease of generation from a clear specification and persistence across intermediate products (such as design or early code modules). Disadvantages include inconsistency as analysts interpret the notional constructs (leading to the need for careful and consistent analyst training) and the difficulty of assessing the size of embedded systems.

- Lack of empirical evidence, especially for new sizing methods. A new sizing method may be more appropriate for a new development technique than are existing methods, but there may not yet be empirical evidence available to suggest appropriate values for input variables.
- Using past project experience and information. Many estimation techniques rely to some degree on the availability of information about past projects. This reliance can leverage lessons learned on earlier projects and reduce variability in input values. However, seeming similarities may mask significant differences in the new project. In addition, historical information may not be in a format useful for a new sizing method.
- Tracking changes and progress over time. Using size to track progress may help to manage the expectations of developers and customers alike. But many sizing models are designed to be used at the beginning of development, not in the middle; a size estimate built from the factors related to one goal may be inappropriate when the goal changes. Moreover, different size measures generated over the course of development may not be comparable over time.
- Calibrating the model. Calibration tailors the model to an organization or development style. When the calibration is performed carefully, the resulting tailored models tend to be more accurate than all-purpose ones. However, new or radically different projects may not be estimated accurately from the calibrated model.

After discussing the ramifications of each issue, we describe seven different sizing methods that the analyst may use (see pp. 23–41). For each method, we present its sources or origins in

software literature, useful references to related web sites and articles, and a description of how each method works, when to use it, and when not to use it. Included are the following:

- Source lines of code (SLOC): a method that estimates the total number of lines of code in the finished software project
- Function points and feature points: methods that measure the amount of functionality in a system by counting and weighting inputs, outputs, queries, and logical and interface files
- Object points: a method that measures size by high-effort items, such as server data tables, client data tables, and screens and reports reused from previous projects
- Application points: a method building on object points, adding rating scales of a project's productivity
- Predictive object points: a method also building on object points, adding information about how objects are grouped into classes
- Analogies: a method using other, completed projects with similar characteristics to the proposed project to suggest the likely size
- Unified Modeling Language (UML) constructs: a relatively new method based on use case, a technique for describing how users will interact with the system to perform functions.

For example, Boehm et al. (2000) revised the object-point approach for use in the COCOMO II estimation process. Calling their technique "application points" to avoid confusion with object points and object-oriented development, they added rating scales to determine a project's productivity in new object points per person-month, the development environment's maturity and capability, and the developer's experience and capability in using the development (integrated, computer-assisted software engineering, or ICASE) environment. That is, application points are an enhancement of object points, designed to include more information about the project and, thus, to reduce uncertainty. A table assists analysts in choosing a rating (from very low to very high) for each of the three additional

scales; the ratings are combined with other ratings. Then the resulting application points measure acts as a size input to an effort estimate. The estimated number of person-months is calculated as the number of application points divided by the productivity measure in the table. Application points are to be used specifically with COCOMO II effort- and schedule-estimation models. There is no evidence that application points are useful in models other than COCOMO II. However, as other estimating techniques embrace the changes in COCOMO II, new evidence may support a decision to switch to application points for sizing.

Of course, all sizing methods have their advantages and disadvantages, depending on the level of knowledge about the system; variation in the languages and structures used to implement the system; and system composition (the use of new, reused, and modified code within a system). Selecting the appropriate size-estimation method helps mitigate the risks associated with each choice.

## Risks in Size Estimation

Risk occurs at many points in a project's life cycle and is tied to activities or to timing. When a decision or choice is made (whether on the micro-level, such as how to design a particular software module or on the macro-level, such as which software architecture to employ), an element of uncertainty is introduced in the estimation process; this choice increases the risk and, thus, the chance for error. This uncertainty is further aggravated when cost estimates must be made very early in the project's life cycle. (See pp. 43–53.)

Thus, it is important to recognize the risks and deal with them properly. One source of estimation error is the presence of incorrect or incomplete data elements, such as descriptions of how the software will be developed or notions of how the user will use the software system. Another source of error derives from correct data being used incorrectly, as when a computation is not complete or is applied inappropriately. But these errors themselves are derived from three

kinds of uncertainty: (1) in the specification or design, (2) about the development method, and (3) in the estimation process.

We consider the following risks important to each of the above categories:

- Uncertainty in the specification or design
  - Problems in understanding the requirements or design
  - Incomplete or inconsistent requirements or design
- Uncertainty about the development method
  - Economies and diseconomies of scale
  - Mismatch between the proposed development method and the estimation's assumed method
- Uncertainty in the estimation process
  - Subjectivity and lack of independence in the adjustment factors
  - Counter-intuitive values for adjustment factors
  - Adjustment factors that seem irrelevant to the current project
  - Rater bias
  - Inter-rater disagreements
  - Inappropriate use of measurement.

Each of these risks is described in terms of symptoms and warning signs; these, in turn, can alert the analyst to the possibility of risk, and we recommend mitigation strategies for each. For example, consider the risk of diseconomies of scale. Sometimes, techniques that have good effects in the small can have bad effects in the large. For instance, using formal methods to prove the correctness of requirements has been shown to find problems in requirements, but using formal methods on a large scale can be expensive, time-consuming, and sometimes infeasible. Symptoms of diseconomies of scale include inability to judge the effects of the candidate technology on the size of development and inability to decide which parts of the system should be subjected to the technology (such as deciding which portions of the requirements should be proven correct using formal methods). To mitigate this risk, it may be useful to decompose the system into subsystems and then do a size estimate for each sub-

system. Such decomposition can be based on the work breakdown structure (WBS, a formal description of the tasks and their dependencies) or on functional subsystems, each of which will be developed in a different way.

In addition to describing each risk, we provide a risk checklist for size estimation to which an analyst may refer repeatedly throughout the project's life cycle. This checklist refers to three important stages in the project life cycle: selection of the sizing method, assessment of the project/system, and application of the cost-estimation method. In each of these stages, we suggest actions that may help the analyst to avoid risks in the short term and long term. (See pp. 55–59.)

## **Approaches to Cost Estimation (see pp. 61–76)**

Sizing is only one aspect of estimating how much effort will be involved in developing, delivering, and maintaining software. We analyze the broader issues of cost estimation, acknowledging that cost estimation is as much an art as a science.

Cost estimates for software development and maintenance activities are frequently associated with decisions about affordability, investment, and value. *Affordability* includes not only the costs necessary to accomplish the development but also those costs that address training, repair, and upgrades over the intended system's life cycle. *Investment* decisions consider whether the associated costs will yield a specific capability within the time and resources available. *Value* may consider whether other options can provide a more affordable or less risky investment to achieve the desired capability.

Thus, the way in which a cost estimate is used often depends on the types of decisions that need to be made, when they are needed, and who is making them. In particular, we can view a cost estimate from the perspective of the system's buyer, developer, or user, as well as from the perspective of a researcher who is trying to analyze how well a model or technique meets intended needs. The different uses of cost estimates suggest that the inherent risks differ, based on perspec-

tive and need. Thus, the relationship of risk to cost estimation can be understood only with a concomitant understanding of how the estimation is performed.

To that end, we review several widely recognized methods for estimating software cost, from informal methods that rely heavily on experience and expertise, to very formal parametric methods based on formulas derived from past performance. The methods include expert judgment, analogy, parametric and algorithmic methods, bottom-up (work breakdown structure) methods, and top-down methods. For each method, we describe how it works, the advantages and disadvantages, and appropriate usage.

For example, methods using analogy rely on data from actual projects, thereby avoiding expert judgment's reliance on recall. They also avoid the complexity of parametric/algorithmic models. Templates can be built to characterize different kinds of projects or project attributes, to explicitly account for differences between previous projects and the proposed project. Tools, such as Bournemouth University's ANGEL (Shepperd and Schofield, 1997), can be used to support the estimation.

However, there are several disadvantages to using analogies. Because this method depends on expert judgment to account for differences and to extrapolate from a previous project to the current project, it can be challenging and subjective. Two projects that may seem similar may indeed be different in a critical way (just as a runner who runs a four-minute mile cannot run a marathon in under two hours). Moreover, the uncertainty in assessing similarity and difference means that two different analysts may have significantly different views and eventual estimates. This difficulty can be mitigated by using historical data, which in turn requires maintaining and using a database of templates or project data.

As with expert judgment, analogy is not suitable when the estimation analysts have neither experience nor data for similar projects. Similarly, the method is not useful when some aspect of the proposed system is dramatically different in some way from most of the other projects in the database or in the analysts' experience. However, analogies may be useful when estimates are needed from sparse, high-

level system descriptions, particularly before detailed design or requirements are fully specified.

Each of the estimation approaches described can be enhanced by the existence and use of a historical database of project information. Not only can models be derived from such data, but the data are also essential for calibrating models, suggesting confidence levels, supporting expert judgments and analogies, and assisting any reality check of an estimate supplied to another source.

However, historical databases are like good hygiene: Everyone acknowledges that they are good to have, but not everyone follows through with careful practice. It takes time and effort to define the appropriate data elements, build a repository, gather and verify data, provide an effective interface to enable analysts to retrieve appropriate data, and use those data to build and calibrate models. In addition, the data may be proprietary or difficult to obtain by those maintaining the database. The costs related to the care and feeding of historical databases must be compared with the cost of generating poor estimates. In almost every case, the investment in historical data is well worth it (Boehm et al., 2000).

## **Risks in Cost Estimation (see pp. 77–89)**

Much as with sizing error, error is introduced into the data and estimation process as a function of three types of uncertainty: in the system definition, in the system development, and in the estimation method. For each type, we analyze the indicators of risk and suggest steps to be taken to address it. For example, during system definition, the problem to be solved may not be well defined. Symptoms may include different interpretations of what is needed, substantial use of “TBD” or “TBS” (to be determined or supplied) in the specification, or constant change to the specification. If the system use is not well understood, the concept of operations may be incomplete, inconsistent, or ambiguous. And, if the system is pushing the limits of technology, key requirements or functions may be included in the program risk plan. To address these risks, the likely cost can be expressed

as a range, not as a point estimate; several estimates can be made over time, and estimation assumptions can be challenged repeatedly.

Similarly, risk is introduced during system development. Uncertainty in the development process is indicated when critical-path activities are unknown or unresolvable, or when there is lack of evidence that the developers are heeding or will adhere to software management plans. Other indicators of uncertainty are lack of consideration of the trade-off between maintaining a component or rebuilding it from scratch; lack of anticipation of potential defects; a mismatch between key personnel's experience and current needs; and lack of information about the consequences of possible loss. These system development risks can be addressed in several ways, including conducting several estimates over time, requiring having details on developing-organization performance, using historical data to support decisionmaking, and reviewing program documentation.

Estimation-process risk is introduced during method selection, application, and interpretation, and it can be addressed at several stages of the estimation process. When methods and tools are selected, warning signs of risk include lack of consideration of system characteristics (such as development approach, complexity, or size), intermediate results inconsistent with analysts' experience or expectations, or a mismatch between model goals and analysts' needs. During data collection, warning signs include insufficient information for use with the estimation model or inconsistent data inputs. When analysts review and evaluate a model's results, an unreasonable picture of likely cost and schedule is a signal that the model has been used improperly or is inappropriate for the situation.

Several steps can be taken to address these risks effectively. They range from garnering as much information as possible about the project to taking time to understand what methods each model uses, and whether those methods/models are appropriate to the project at hand. In addition, the developers can ensure that staff is trained on how to use each model. Cost analysts must be able to understand and document all the assumptions of their selected estimation model.

Well-trained cost analysts can generate reasonable estimates for each model variable, preferably in advance of examining the con-

tractor data. If possible, they can conduct a sensitivity analysis on those key variables that engender significant uncertainty. Where possible, analysts can use multiple models to “triangulate” a reasonable estimate. In addition, they can verify reasonableness with expert judgment.

Other risk-reduction measures include using correct and appropriate economic data, such as cost rates for personnel, to support each model input. Analysts should pay careful attention to the scale or units required for each variable, such as constant dollars or dollars adjusted for inflation. In addition, they should understand whether and how each method or model considers maintenance costs and time, and adjust accordingly. Wherever possible, analysts can simplify models by concentrating on using inputs with the most effect and eliminating inputs that have very little effect on the resulting estimate. The effect of each input can be assessed retrospectively by performing a sensitivity analysis on each input; those inputs whose differences yield little change in the overall estimate can be eliminated in future estimates.

By developing and retaining a repository of historical data and metrics, cost analysts can use the data to support realistic inputs, to check the realism of outputs, and to provide feedback, learning, and comparison.

## **Final Directions (see pp. 91–92)**

The information provided in this report can be used in two ways: to address techniques for improving current estimation methods and to find new methods when existing ones prove inadequate. The latter function is particularly important. Software development is changing, reflecting not only the need to find better ways to build better software but also the market pressures to use different technologies as they are proposed by the research and commercial communities.

An inaccurate estimate does not always mean a bad estimating technique or an incapable analyst. Instead, it may mean that the technique must be calibrated or extended, or that the analyst may

need refresher training. In the end, the combination of method selection and analyst's action helps to mitigate avoidable risks in estimating software size and cost.