

# Project management

From Wikipedia, the free encyclopedia

**Project management** is the [discipline](#) of organizing and managing resources in such a way that these resources deliver all the [work](#) required to complete a project within defined scope, time, and cost constraints. A [project](#) is a temporary and one-time endeavor undertaken to create a unique product or service. This property of being a temporary and a one-time undertaking contrasts with [processes](#), or operations, which are permanent or semi-permanent ongoing functional work to create the same product or service over-and-over again. The management of these two systems is often very different and requires varying technical skills and philosophy, hence requiring the development of project management.

The first challenge of project management is ensuring that a project is delivered within the defined constraints. The second, more ambitious, challenge is the [optimized allocation](#) and integration of the inputs needed to meet those pre-defined objectives. The [project](#), therefore, is a carefully selected set of activities chosen to use [resources](#) ([time](#), [money](#), [people](#), [materials](#), [energy](#), [space](#), [provisions](#), [communication](#), [quality](#), [risk](#), etc.) to meet the pre-defined objectives.

## Contents

[\[hide\]](#)

- [1 The project manager](#)
- [2 The traditional triple constraints](#)
  - [2.1 Time](#)
  - [2.2 Cost](#)
  - [2.3 Scope](#)
- [3 Project Management activities](#)
- [4 Project Management artifacts](#)
- [5 Project control variables](#)
- [6 History of Project Management](#)
- [7 Approaches](#)
  - [7.1 The traditional approach](#)
  - [7.2 Critical chain](#)
  - [7.3 Process-based management](#)

- [8 Project systems](#)
  - [8.1 Project control systems](#)
  - [8.2 Project development stages](#)
    - [8.2.1 Initiation](#)
    - [8.2.2 Planning and design](#)
    - [8.2.3 Production or execution](#)
    - [8.2.4 Closing and Maintenance](#)
- [9 Project Management Associations](#)
  - [9.1 International Standards](#)
  - [9.2 Professional Certifications](#)
- [10 See also](#)
- [11 Literature](#)
- [12 External links](#)

## The project manager

Project management is quite often the province and responsibility of an individual [project manager](#). This individual seldom participates directly in the activities that produce the end result, but rather strives to maintain the progress and productive mutual interaction of various parties in such a way that overall risk of failure is reduced.

A project manager is often a client representative and has to determine and implement the exact needs of the client based on knowledge of the firm he/she is representing. The ability to adapt to the various internal procedures of the contracting party, and to form close links with the nominated representatives, is essential in ensuring that the key issues of cost, time, quality and above all, client satisfaction, can be realized.

In whatever field, a successful project manager must be able to envisage the entire project from start to finish and to have the ability to ensure that this vision is realized.

Any type of product or service - buildings, vehicles, electronics, computer software, financial services, etc. - may have its implementation overseen by a project manager and its operations by a product manager.

## The traditional triple constraints

Like any human undertaking, projects need to be performed and delivered under certain constraints. Traditionally, these constraints have been listed as: scope, time, and cost. This is also referred to as the Project Management Triangle where each side represents a constraint. One side of the triangle cannot be changed without impacting the others. A further refinement of the constraints separates product 'quality' or 'performance' from scope, and turns quality into a fourth constraint.

The time constraint refers to the amount of time available to complete a project. The cost constraint refers to the budgeted amount available for the project. The scope constraint refers to what must be done to produce the project's end result. These three constraints are often competing constraints: increased scope typically means increased time and increased cost, a tight time constraint could mean increased costs and reduced scope, and a tight budget could mean increased time and reduced scope.

The discipline of project management is about providing the tools and techniques that enable the project team (not just the project manager) to organize their work to meet these constraints.

### Time

Broken down for analytical purposes into the time required to complete the components of the project, which is then further broken down into the time required to complete each task contributing to the completion of each component. When performing tasks using project management, it is important to cut the work into smaller pieces so that it is easy to follow.

### Cost

Cost to develop a project depends on several variables including (chiefly): labor rates, material rates, [risk management](#), plant (buildings, machines, etc.), equipment, and profit. When hiring an independent consultant for a project, cost will typically be determined by the consultant's or firm's [per diem](#) rate multiplied by an estimated quantity for completion.

### Scope

Requirements specified for the end result. The overall definition of what the project is supposed to accomplish and a specific description of what the end result should be or accomplish. A major component of scope is the [quality](#) of the final product. The amount of time put into individual tasks determines the overall quality of the project. Some tasks may require a given amount of time to complete adequately, but given more time could be completed exceptionally. Over the course of a large project, quality can have a significant impact on time and cost (or vice versa).

## Project Management activities

Project Management is composed of several different types of activities such as:

1. Planning the work or objectives
2. Analysis & Design of objectives
3. Assessing and mitigating risk (or [Risk Management](#))
4. Estimating resources
5. Allocation of resources
6. Organizing the work
7. Acquiring human and material resources
8. Assigning tasks
9. Directing activities
10. Controlling project execution
11. Tracking and Reporting progress
12. Analyzing the results based on the facts achieved
13. Defining the products of the project
14. Forecasting future trends in the project
15. Quality Management
16. Issues Management

## Project Management artifacts

All successful projects adequately document objectives and deliverables. These documents are a mechanism to align sponsors, clients, and project team's expectations.

1. Project Charter
2. [Business case](#)/Feasibility Study
3. [Scope Statement](#) / [Terms of reference](#)
4. [Project Management Plan](#) / Project Initiation Document
5. [Work Breakdown Structure](#)
6. Change Control Plan
7. [Risk Management Plan](#)
8. Communications Plan
9. Governance Model

10. [Risk Register](#)
11. Issue Log
12. Action Item List
13. Resource Management Plan
14. [Project Schedule](#)
15. Status Report
16. [Responsibility assignment matrix](#)
17. Database of risks
18. Database of lessons learned
19. Stakeholder Analysis

These documents are normally hosted on a shared resource (i.e., Intranet web page) and are available for review by the project's stakeholders. Changes or updates to these documents are explicitly outlined in the project's configuration management (or change control plan).

## Project control variables

Project Management tries to gain control over variables such as risk:

### [risk](#)

Potential points of failure. Most negative risks (or potential failures) can be overcome or resolved, given enough planning capabilities, time, and resources. According to some definitions (including PMBOK Third Edition) risk can also be categorized as "positive--" meaning that there is a potential opportunity, e.g., complete the project faster than expected.

Customers (either internal or external project sponsors), external organizations (such as government agencies and regulators) can dictate the extent of three variables: time, cost, and scope. The remaining variable (risk) is managed by the project team, ideally based on solid estimation and response planning techniques. Through a negotiation process among project stakeholders, an agreement defines the final objectives, in terms of time, cost, scope, and risk, usually in the form of a charter or contract.

To properly control these variables a good project manager has a depth of knowledge and experience in these four areas (time, cost, scope, and risk), and in six other areas as well: integration, communication, human resources, quality assurance, schedule development, and procurement.

## History of Project Management

As a discipline, Project Management developed from several different fields of application, including construction, mechanical engineering, military projects, etc. In the United States, the forefather of project management is [Henry Gantt](#), called the father of planning and control techniques, who is famously known for his use of the "[bar](#)" [chart](#) as a project management tool, for being an associate of [Frederick Winslow Taylor](#)'s theories of [scientific management](#), and for his study of the work and management of Navy ship building. His work is the forerunner to many modern project management tools, including the work breakdown structure (WBS) and resource allocation.

The 1950's mark the beginning of the modern project management era. Again, in the United States, prior to the 1950's, projects were managed on an ad hoc basis using mostly [Gantt Charts](#), and informal techniques and tools. At that time, two mathematical project scheduling models were developed: (1) the "[Program Evaluation and Review Technique](#)" or PERT, developed as part of the [United States Navy](#)'s (in conjunction with the [Lockheed Corporation](#)) [Polaris missile](#) submarine program; and (2) the "[Critical Path Method](#)" (CPM) developed in a joint venture by both [DuPont Corporation](#) and [Remington Rand Corporation](#) for managing plant maintenance projects. These mathematical techniques quickly spread into many private enterprises.

In 1969, the [Project Management Institute](#) (PMI) was formed to serve the interest of the project management industry. The premise of PMI is that the tools and techniques of project management are common even among the widespread application of projects from the [software industry](#) to the construction industry. In 1981, the PMI Board of Directors authorized the development of what has become the *The Guide to the Project Management Body of Knowledge*, containing the standards and guidelines of practice that are widely used throughout the profession.

## Approaches

There are several approaches that can be taken to managing project activities including agile, interactive, incremental, and phased approaches.

Regardless of the approach employed, careful consideration needs to be given to clarify surrounding project objectives, goals, and importantly, the roles and responsibilities of all participants and stakeholders.

### *The traditional approach*

A traditional phased approach identifies a sequence of steps to be completed. In the **traditional approach**, we can distinguish 5 components of a project (4 stages plus control) in the development of a project:

1. project initiation stage;
2. [project planning](#) or design stage;
3. project execution or production stage;

4. project monitoring and controlling systems;
5. project completion stage.

Not all the projects will visit every stage as projects can be terminated before they reach completion. Some projects probably don't have the planning and/or the monitoring. Some projects will go through steps 2, 3 and 4 multiple times.

Many industries utilize variations on these stages. For example, in bricks and mortar architectural design, projects typically progress through stages like Pre-Planning, Conceptual Design, Schematic Design, Design Development, Construction Drawings (or Contract Documents), and Construction Administration. In [software development](#), this approach is often known as 'waterfall development' i.e one series of tasks after another in linear sequence. In software development many organizations have adapted the Rational Unified Process (RUP) to fit this methodology, although RUP does not require or explicitly recommend this practice. Waterfall development can work for small tightly defined projects, but for larger projects of undefined or unknowable scope, it is less suited. Because software development is often the realization of a new or novel product, this method has been widely accepted as ineffective for software projects where requirements a largely unknowable up front and susceptible to change. While the names may differ from industry to industry, the actual stages typically follow common steps to [problem solving](#)--*defining the problem, weighing options, choosing a path, implementation and evaluation*.

## **Critical chain**

[Critical chain](#) is an extension to the traditional [critical path](#) method.

In critical studies of project management, it has been noted that several of these fundamentally [PERT](#)-based models are not well suited for the multi-project company environment of today. Most of them are aimed at very large-scale, one-time, non-routine projects, and nowadays all kinds of management are expressed in terms of projects. Using complex models for "projects" (or rather "tasks") spanning a few weeks has been proven to cause unnecessary costs and low maneuverability in several cases. Instead, project management experts try to identify different "lightweight" models, such as, for example [Extreme Programming](#) for software development and [Scrum](#) techniques. The generalization of Extreme Programming to other kinds of projects is [extreme project management](#), which may be used in combination with the [process modeling](#) and management principles of [human interaction management](#).

## **Process-based management**

Also furthering the concept of project control is the incorporation of [process-based management](#). This area has been driven by the use of Maturity models such as the [CMMI](#) (Capability Maturity Model Integration) and [ISO/IEC15504](#) (SPICE - Software Process Improvement and Capability Determination), which have been far more successful.

Agile project management approaches based on the principles of [human interaction management](#) are founded on a process view of human collaboration. This contrasts sharply with traditional approach. In the [agile software development](#) or [flexible product development](#) approach, the project is seen as a series of relatively small tasks conceived and executed as the situation demands in an adaptive manner, rather than as a completely pre-planned process.

## Project systems

As mentioned above, traditionally, project development includes five elements: control systems and four stages.

### Project control systems

Project control is that element of a project that keeps it on-track, on-time, and within budget. Project control begins early in the project with planning and ends late in the project with post-implementation review, having a thorough involvement of each step in the process. Each project should be assessed for the appropriate level of control needed, too much control is too time consuming, too little control is too costly. Clarifying the cost to the business if the control is not implemented in terms of errors, fixes, and additional [audit](#) fees.

Control systems are needed for cost, [risk](#), quality, communication, time, change, procurement, and human resources. In addition, [auditors](#) should consider how important the projects are to the [financial statements](#), how reliant the stakeholders are on controls, and how many controls exist. [Auditors](#) should review the development process and procedures how they are implemented. The process of development and the quality of the final product may also be assessed if needed or requested. A business may want the [auditing](#) firm to be involved throughout the process to catch problems earlier on so that they can be fixed more easily. An [auditor](#) can serve as a controls [consultant](#) as part of the development team or as an independent [auditor](#) as part of an [audit](#).

Businesses sometimes use formal systems development processes. These help assure that systems are developed successfully. A formal process is more effective in creating strong controls, and [auditors](#) should review this process to confirm that it is well designed and is followed in practice. A good formal systems development plan outlines:

- A [strategy](#) to align development with the organization's broader objectives
- Standards for new systems
- Project management policies for timing and [budgeting](#)
- Procedures describing the process

### Project development stages

Regardless of the methodology used, the project development process will have the same major stages: initiation, development, production or execution, and closing/maintenance.



## Initiation

The **initiation** stage determines the nature and scope of the development. If this stage is not performed well, it is unlikely that the project will be successful in meeting the business's needs. The key project controls needed here is an understanding of the business environment and making sure that all necessary controls are incorporated into the project. Any deficiencies should be reported and a recommendation should be made to fix them.

The initiation stage should include a cohesive plan that encompasses the following areas:

- Study analyzing the business needs in measurable goals.
- Review of the current operations.
- Conceptual design of the operation of the final product.
- Equipment requirement.
- [Financial analysis](#) of the costs and benefits including a [budget](#).
- Select stake holders, including users, and support personnel for the project.
- Project charter including costs, tasks, deliverables, and schedule.

## Planning and design

After the initiation stage, the system is designed. Occasionally, a small prototype of the final product is built and tested. Testing is generally performed by a combination of testers and end users, and can occur after the prototype is built or concurrently. Controls should be in place that ensure that the final product will meet the specifications of the project charter. The results of the design stage should include a product design that:

- Satisfies the project sponsor, end user, and business requirements.
- Functions as it was intended.
- Can be produced within quality standards.
- Can be produced within time and budget constraints.

## Production or execution

The execution stage includes the actual implementation of the design or plan. In software systems, this includes conversion (transfer of data from an old system to a new system), documentation, and training. From an [auditor's](#) perspective, training is also important because it helps users use the [software](#) correctly. The bulk of the project's work and largest capital expenditure is realized in this stage.

## Closing and Maintenance

Closing includes the formal acceptance of the project and the ending thereof. Administrative activities include the archiving of the files and documenting lessons learned.

Maintenance is an ongoing process, and it includes:

- Continuing support of end users
- Correction of errors
- Updates of the [software](#) over time

In this stage, [auditors](#) should pay attention to how effectively and quickly user problems are resolved.

Over the course of any construction project, the work scope changes. Change is a normal and expected part of the construction process. Changes can be the result of necessary design modifications, differing site conditions, material availability, contractor-requested changes, value engineering and impacts from third parties, to name a few. Beyond executing the change in the field, the change normally needs to be documented to show what was actually constructed. Hence, the owner usually requires a final record to show all changes or, more specifically, any change that modifies the tangible portions of the finished work. The record is made on the contract documents – usually, but not necessarily limited to, the design drawings. The end product of this effort is what the industry terms as-built drawings, or more simply, “asbuilts.” The requirement for providing them is a norm in construction contracts.

## Project Management Associations

Several national and professional associations exist which has as their aim the promotion and development of project management and the project management profession. The most prominent associations include:

- The [Association for Project Management](#) (UK) ([APM](#))
- The [Australian Institute of Project Management](#) ([AIPM](#))
- The [International Project Management Association](#) ([IPMA](#))
- The [Project Management Institute](#) ([PMI](#))
- [The International Association of Project and Program Management](#) ([IAPPM](#))
- The [International Project Management Commission](#) ([IPMC](#))

## International Standards

- [A Guide to the Project Management Body of Knowledge](#) (*PMBOK Guide*)

- [\*APM Body of Knowledge 5th ed.\*](#) (*APM - Association for Project Management (UK)*)
- [PRINCE2](#) (*PRojects IN a Controlled Environment*)
- [P2M](#) (*A guidebook of Project & Program Management for Enterprise Innovation*, Japanese third-generation project management method)
- [V-Modell](#) (German project management method)
- [HERMES](#) (The Swiss general project management method, selected for use in Luxembourg and international organisations)
- [OPM3](#)

## Professional Certifications

There have been several attempts to develop project management [standards](#), such as:

- [ISO 10006:1997](#), *Quality management - Guidelines to quality in project management*
- [CPM](#) ([The International Association of Project & Program Management])
- [ISEB Project Management Syllabus](#)
- [JPACE](#) (Justify, Plan, Activate, Control, and End - The James Martin Method for Managing Projects (1981-present))
- [Project Management Professional](#), [Certified Associate in Project Management](#). [PMI](#) certifications

See also: [An exhaustive list of standards \(maturity models\)](#)

So far, there is no known attempt to develop a project management standard available under the [GNU Free Documentation License](#). There was a proposed [Project Management XML Schema](#).

There is an effort by PMI to develop [The Practice Standard for Scheduling](#). (This document is currently (May 2006) in exposure draft, which is near the end of the [standards development process](#).)

## See also

- [Architecture](#)
- [Architectural engineering](#)
- [Association for Project Management](#)
- [Building engineering](#)
- [Building construction](#)
- [Capability Maturity Model](#)
- [Commonware](#)
- [Construction engineering](#)

- [Construction management](#)
- [Construction software](#)
- [Cost overrun](#)
- [Critical chain](#)
- [Critical path method](#)
- [Dependency Structure Matrix](#)
- [Earned value management](#)
- [EuroMPM - European Master in Project Management](#)
- [Estimation](#)
- [Flexible project management](#)
- Functionality, mission and scope [creep](#)
- [Gantt chart](#)
- [Governance](#)
- [Human Interaction Management](#)
- [Industrial Engineering](#)
- [List of project management topics](#)
- [Management](#)
- [Megaprojects](#)
- [Metrics](#)
- [M.S.P.M.](#)
- [Portfolio management](#)
- [Project accounting](#)
- [Program management](#)
- [Project management software \(List of project management software\)](#)
- [Project management in the building process chain](#)
- [Process architecture](#)
- [RACI diagram](#)
- [Software project management](#)
- [Terms of reference](#)
- [The Mythical Man-Month](#)
- [Timesheet](#)
- [Work Breakdown Structure](#)

## Literature

- Berkun, Scott (2005). *Art of Project Management*. Cambridge, MA: O'Reilly Media. [ISBN 0-596-00786-8](#).
- Brooks, Fred (1995). *The Mythical Man-Month*, 20th Anniversary Edition, Adison Wesley. [ISBN 0-201-83595-9](#).
- Heerkens, Gary (2001). *Project Management (The Briefcase Book Series)*. McGraw-Hill. [ISBN 0-07-137952-5](#).
- Kerzner, Harold (2003). *Project Management: A Systems Approach to Planning, Scheduling, and Controlling*, 8th Ed., Wiley. [ISBN 0-471-22577-0](#).
- Lewis, James (2002). *Fundamentals of Project Management*, 2nd ed., American Management Association. [ISBN 0-8144-7132-3](#).
- Meredith, Jack R. and Mantel, Samuel J. (2002). *Project Management : A Managerial Approach*, 5th ed., Wiley. [ISBN 0-471-07323-7](#).
- Project Management Institute (2003). *A Guide To The Project Management Body Of Knowledge*, 3rd ed., Project Management Institute. [ISBN 1-930699-45-X](#).
- Stelman, Andrew and Greene, Jennifer (2005). *Applied Software Project Management*. Cambridge, MA: O'Reilly Media. [ISBN 0-596-00948-8](#).
- Thayer, Richard H. and Yourdon, Edward (2000). *Software Engineering Project Management*, 2nd Ed., Wiley-IEEE Computer Society Press. [ISBN 0-8186-8000-8](#).
- Whitty, Stephen Jonathan (2005). *A Memetic Paradigm of Project Management*. International Journal of Project Management, 23 (8) 575-583.
- Pettee, Stephen R. (2005). *As-built – Problems & Proposed Solutions*. Construction Management Association of America.
- Verzuh, Eric (2005). *The Fast Forward MBA in Project Management*, 2nd, Wiley. [ISBN 0-471-69284-0](#) (pbk.).

## External links



[Wikiquote](#) has a collection of quotations related to:

### [Project management](#)

- [American Academy of Project Management](#)
- [Association for Project Management](#)
- [History of Project Management](#)
- [Interactive Project Management Wiki](#)
- [International Research Network on Organizing by Projects](#)
- [International Project Management Association](#)
- [International Project Management Commission](#)
- [Project Management Association of America](#)
- [The Australian Institute of Project Management](#)
- [The Institute of Project Management of Ireland](#)
- [The Project Management Institute](#)
- [History of Project Management](#)

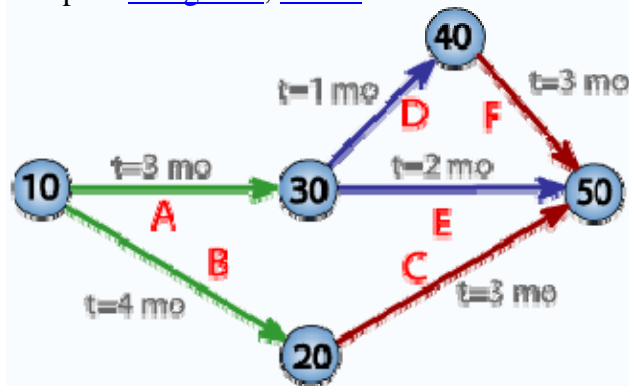
Retrieved from "[http://en.wikipedia.org/wiki/Project\\_management](http://en.wikipedia.org/wiki/Project_management)"


# Program Evaluation and Review Technique

From Wikipedia, the free encyclopedia

(Redirected from [PERT](#))

Jump to: [navigation](#), [search](#)



 PERT network chart for a seven-month project with five milestones (10 through 50) and six activities (A through F).

The **Program Evaluation and Review Technique** commonly abbreviated **PERT** is a model for [project management](#) invented by [Booz Allen Hamilton, Inc.](#) under contract to the [United States Department of Defense](#)'s US Navy Special Projects Office in [1958](#) as part of the [Polaris](#) mobile [submarine](#)-launched ballistic missile project. This project was a direct response to the [Sputnik crisis](#).

PERT is basically a method to analyze the tasks involved in completing a given [project](#), especially the time needed to complete each task, and identifying the minimum time needed to complete the total project.

It was developed in the '50's, primarily to simplify the planning and scheduling of large and complex projects.

It was able to incorporate uncertainty in the sense that it was possible to schedule a project not knowing precisely the details and durations of all the activities.

More of an event-oriented technique rather than start- and completion-oriented.

This technique is used more in R&D-type projects where Cost is not a major factor but Time is.

This project model was the first of its kind, a revival for [scientific management](#), founded in [Fordism](#) and [Taylorism](#). Though every company now has its own "project model" of some kind, they all resemble PERT in some respect. Only [DuPont corporation's critical path method](#) was invented at roughly the same time as PERT.

The most famous part of PERT is the "PERT Networks", charts of timelines that interconnect. PERT is intended for very large-scale, one-time, complex, non-routine projects.

## Contents

[\[hide\]](#)

- [1 PERT terminology and conventions](#)
  - [1.1 Conventions](#)
  - [1.2 Terminology](#)
- [2 Implementing PERT](#)
- [3 See also](#)
- [4 External links](#)

## PERT terminology and conventions

### Conventions

- A **PERT chart** is a tool that **facilitates** decision making; The first draft of a PERT chart will number its events sequentially in 10s (10, 20, 30, etc.) to allow the later insertion of additional events.
- Two consecutive events in a PERT chart are linked by **activities**, which are conventionally represented as arrows in the diagram above.
- The events are presented in a logical sequence and no activity can commence until its immediately preceding event is completed.
- The planner decides which milestones should be PERT events and also decides their “proper” sequence.
- A PERT chart may have multiple pages with many sub-tasks.



## Terminology

- A *PERT event*: is a point that marks the start or completion of one (or more) tasks. It consumes **no time**, and uses **no resources**. It marks the completion of one (or more) tasks is not “reached” until **all** of the activities leading to that event have been completed.
- A *predecessor event*: an event (or events) that immediately precedes some other event without any other events intervening. It may be the consequence of more than one activity.
- A *successor event*: an event (or events) that immediately follows some other event without any other events intervening. It may be the consequence of more than one activity.
- A *PERT activity*: is the actual performance of a task. It consumes **time**, it requires **resources** (such as labour, materials, space, machinery), and it can be understood as representing the time, effort, and resources required to move from one event to another. A PERT activity cannot be completed until the event preceding it has occurred.
- *Optimistic time (O)*: the minimum possible time required to accomplish a task, assuming everything proceeds better than is normally expected
- *Pessimistic time (P)*: the maximum possible time required to accomplish a task, assuming everything goes wrong (but excluding major catastrophes).
- *Most likely time (M)*: the best estimate of the time required to accomplish a task, assuming everything proceeds as normal.
- *Expected time (T<sub>E</sub>)*: the best estimate of the time required to accomplish a task, assuming everything proceeds as normal (the implication being that the expected time is the average time the task would require if the task were repeated on a number of occasions over an extended period of time).

$$T_E = (O + 4M + P) \div 6$$

- *Critical Path*: the longest possible continuous pathway taken from the initial event to the terminal event. It determines the total calendar time required for the project; and, therefore, any time delays along the critical path will delay the reaching of the terminal event by at least the same amount.
- *Lead time*: the time by which a *predecessor event* must be completed in order to allow sufficient time for the activities that must elapse before a specific PERT event is reached to be completed.
- *Lag time*: the earliest time by which a *successor event* can follow a specific PERT event.
- *Slack*: the **slack** of an event is a measure of the excess time and resources available in achieving this event. **Positive slack** would indicate *ahead of schedule*; **negative slack** would indicate *behind schedule*; and **zero slack** would indicate *on schedule*.

## Implementing PERT

The first step to scheduling the project is to determine the tasks that the project requires and the order in which they must be completed. The order may be easy to record for some tasks (*i.e.* When building a house, the land must be graded before the foundation can be laid) while difficult for others (There are two areas that need to be graded, but there are only enough bulldozers to do one). Additionally, the time estimates usually reflect the normal, non-rushed time. Many times, the time required to execute the task can be [reduced](#) for an additional cost or a reduction in the quality.

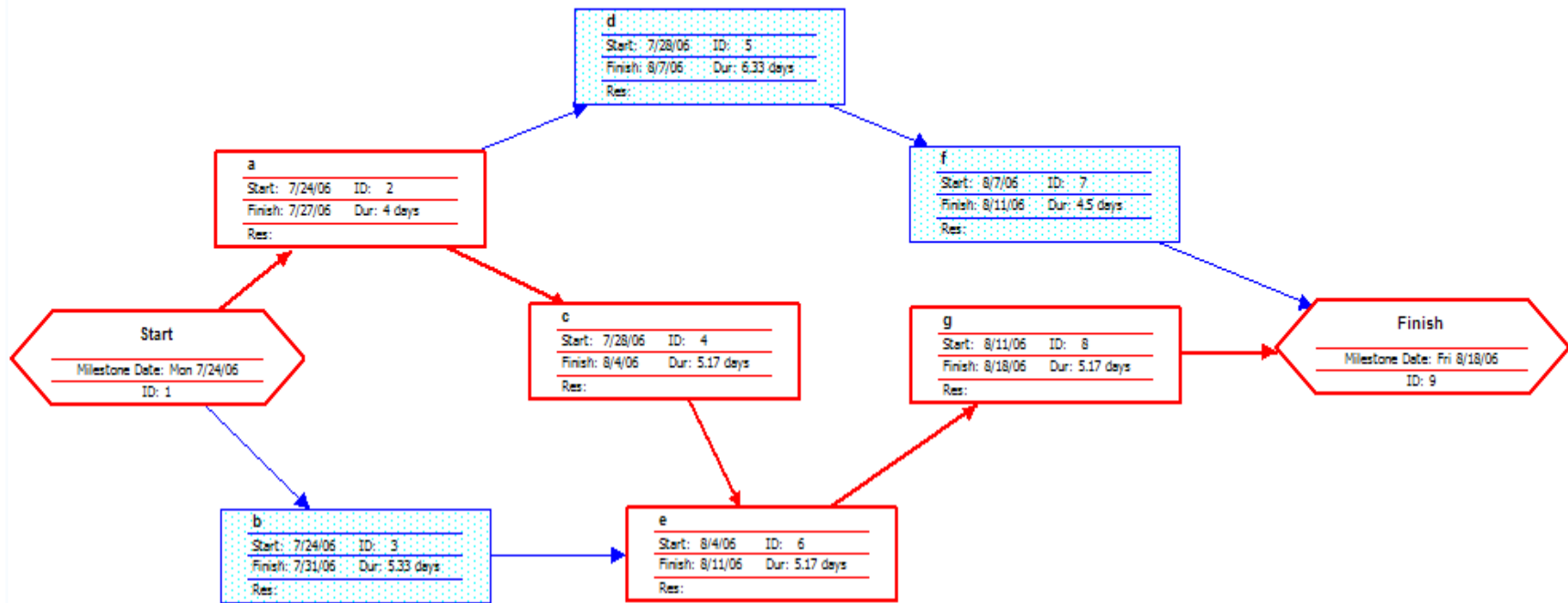
In the following example there are seven tasks, labeled *a* through *g*. Some tasks can be done concurrently (*a* & *b*) while others cannot be done until their predecessor task is complete (*c* cannot begin until *a* is complete). Additionally, each task has three time estimates: the optimistic time estimate (*a*), the most likely or normal time estimate (*m*), and the pessimistic time estimate (*b*). The expected time ( $T_E$ ) is computed using the formula  $(a + 4m + b)/6$ .

Activity	Predecessor	Opt. a	Norm. m	Pess. b	$T_E$ $(a + 4m + b)/6$
a	--	2	4	6	4.00
b	--	3	5	9	5.33
c	a	4	5	7	5.17
d	a	4	6	10	6.33
e	b, c	4	5	7	5.17
f	d	3	4	8	4.50
g	e	3	5	8	5.17

*Note:* All times listed are in **work days** (Mon - Fri, 8 A.M. to 5 P.M. with a one hour lunch break).

Once this step is complete, one can draw a [Gantt chart](#) or a [network diagram](#).





A network diagram created using [Microsoft Project](#) (MSP). Note the [critical path](#) is in red.

Early Start	Duration	Early Finish
Task Name		
Late Start	Slack	Late Finish

A node like this one (from [Microsoft Visio](#)) can be used to display the activity name, duration, ES, EF, LS, LF, and slack.

By itself, the network diagram pictured above does not give much more information than a Gantt chart; however, it can be expanded to display more information. The most common information shown is:

1. The activity name
2. The normal duration time
3. The early start time (ES)
4. The early finish time (EF)
5. The late start time (LS)
6. The late finish time (LF)
7. The [slack](#)

In order to determine this information it is assumed that the activities and normal duration times are given. The first step is to determine the ES and EF. The ES is defined as the maximum EF of all predecessor activities, unless the activity in question is the first activity, which the ES is zero (0). The EF is the ES plus the task duration ( $EF = ES + \text{duration}$ ).

- The ES for *start* is zero since it is the first activity. Since the duration is zero, the EF is also zero. This EF is used as the ES for *a* and *b*.
- The ES for *a* is zero. The duration (4 work days) is added to the ES to get an EF of four. This EF is used as the ES for *c* and *d*.
- The ES for *b* is zero. The duration (5.33 work days) is added to the ES to get an EF of 5.33.
- The ES for *c* is four. The duration (5.17 work days) is added to the ES to get an EF of 9.17.
- The ES for *d* is four. The duration (6.33 work days) is added to the ES to get an EF of 10.33. This EF is used as the ES for *f*.
- The ES for *e* is the greatest EF of its predecessor activities (*b* and *c*). Since *b* has an EF of 5.33 and *c* has an EF of 9.17, the ES of *e* is 9.17. The duration (5.17 work days) is added to the ES to get an EF of 14.34. This EF is used as the ES for *g*.
- The ES for *f* is 10.33. The duration (4.5 work days) is added to the ES to get an EF of 14.83.

- The ES for *g* is 14.34. The duration (5.17 work days) is added to the ES to get an EF of 19.51.
- The ES for *finish* is the greatest EF of its predecessor activities (*f* and *g*). Since *f* has an EF of 14.83 and *g* has an EF of 19.51, the ES of *finish* is 19.51. *Finish* is a milestone (and therefore has a duration of zero), so the EF is also 19.51.

Barring any [unforeseen events](#), the project should take 19.51 work days to complete. The next step is to determine the late start (LS) and late finish (LF) of each activity. This will eventually show if there are activities that have [slack](#). The LF is defined as the minimum LS of all successor activities, unless the activity is the last activity, for which the LF equals the EF. The LS is the LF minus the task duration ( $LS = LF - \text{duration}$ ).

- The LF for *finish* is equal to the EF (19.51 work days) since it is the last activity in the project. Since the duration is zero, the LS is also 19.51 work days. This will be used as the LF for *f* and *g*.
- The LF for *g* is 19.51 work days. The duration (5.17 work days) is subtracted from the LF to get a LS of 14.34 work days. This will be used as the LF for *e*.
- The LF for *f* is 19.51 work days. The duration (4.5 work days) is subtracted from the LF to get a LS of 15.01 work days. This will be used as the LF for *d*.
- The LF for *e* is 14.34 work days. The duration (5.17 work days) is subtracted from the LF to get a LS of 9.17 work days. This will be used as the LF for *b* and *c*.
- The LF for *d* is 15.01 work days. The duration (6.33 work days) is subtracted from the LF to get a LS of 8.68 work days.
- The LF for *c* is 9.17 work days. The duration (5.17 work days) is subtracted from the LF to get a LS of 4 work days.
- The LF for *b* is 9.17 work days. The duration (5.33 work days) is subtracted from the LF to get a LS of 3.84 work days.
- The LF for *a* is the minimum LS of its successor activities. Since *c* has a LS of 4 work days and *d* has a LS of 8.68 work days, the LF for *a* is 4 work days. The duration (4 work days) is subtracted from the LF to get a LS of 0 work days.
- The LF for *start* is the minimum LS of its successor activities. Since *a* has a LS of 0 work days and *b* has a LS of 3.84 work days, the LS is 0 work days.

The next step is to determine the [critical path](#) and if any activities have [slack](#). The critical path is the path that takes the **longest** to complete. To determine the path times, add the task durations for all available paths. Activities that have slack can be delayed without changing the overall time of the project. Slack is computed in one of two ways,  $\text{slack} = LF - EF$  or  $\text{slack} = LS - ES$ . Activities that are on the critical path have a slack of zero (0).

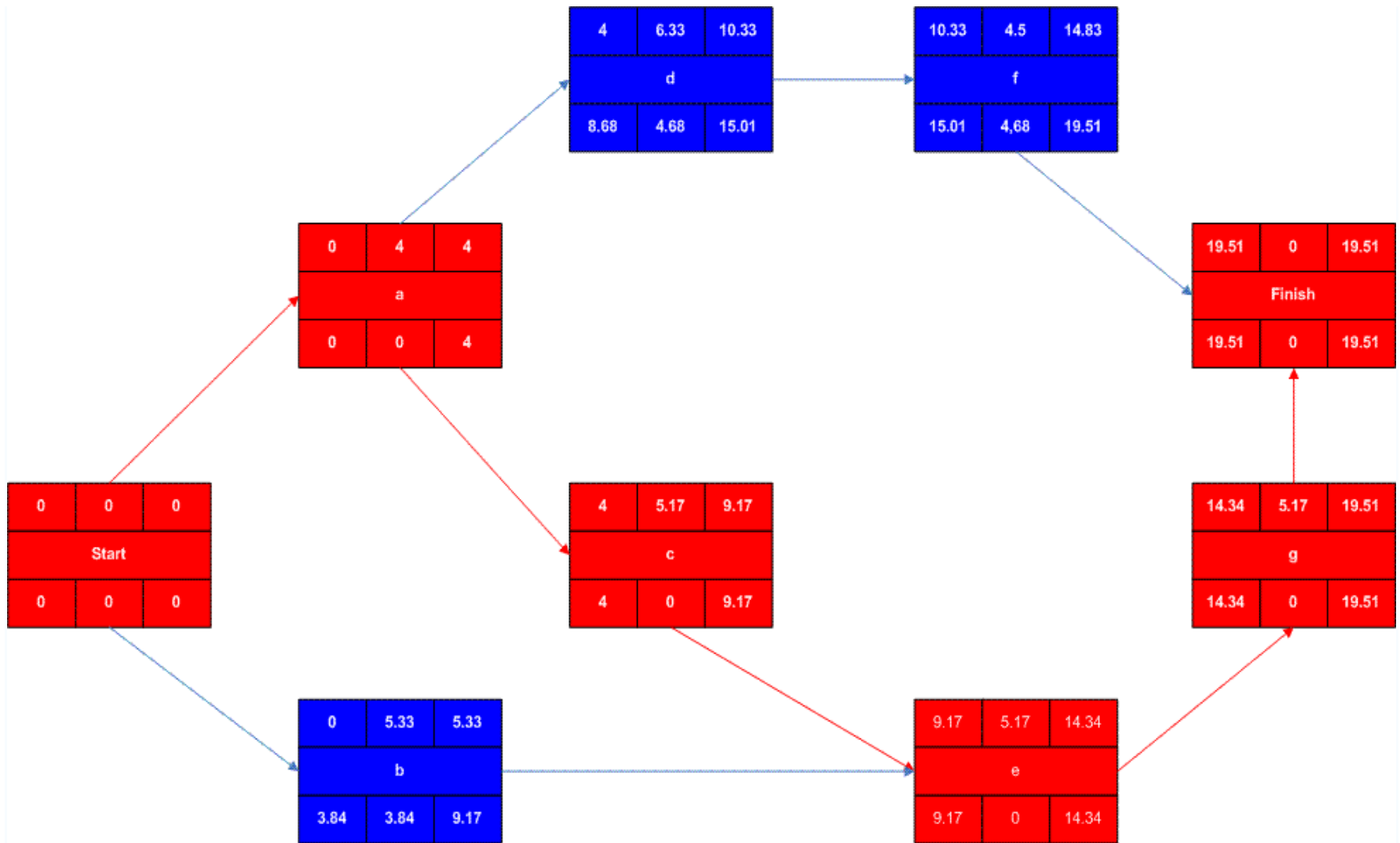
- The duration of path *adf* is 14.83 work days.
- The duration of path *aceg* is 19.51 work days.
- The duration of path *beg* is 15.67 work days.

The critical path is *aceg* and the critical time is 19.51 work days. It is important to note that there can be more than one critical path (in a project more complex than this example) or the critical path can change. For example, let's say that activities *d* and *f* take their pessimistic (b) times to complete instead of their expected ( $T_E$ ) times. The critical path is now *adf* and the critical time is 22 work days. On the other hand, if activity *c* can be [crashed](#) to one work day, the path time for *aceg* is reduced to 15.34 work days, which is slightly less than the time of the new critical path, *beg* (15.67 work days).

Assuming these scenarios do not happen, the slack for each activity can now be determined.

- *Start* and *finish* are milestones and by definition have no duration, therefore they can have no slack (0 work days).
- The activities on the critical path by definition have a slack of zero; however, it is always a good idea to check the math anyway when drawing by hand.
  - $LF_a - EF_a = 4 - 4 = 0$
  - $LF_c - EF_c = 9.17 - 9.17 = 0$
  - $LF_e - EF_e = 14.34 - 14.34 = 0$
  - $LF_g - EF_g = 19.51 - 19.51 = 0$
- Activity *b* has a LF of 9.17 and a EF of 5.33, so the slack is 3.84 work days.
- Activity *d* has a LF of 15.01 and a EF of 10.33, so the slack is 4.68 work days.
- Activity *f* has a LF of 19.51 and a EF of 14.83, so the slack is 3.84 work days.

Therefore, activity *b* can be delayed almost 4 work days without delaying the project. Likewise, activity *d* **or** activity *f* can be delayed 4.68 work days without delaying the project (alternatively, *d* and *f* can be delayed 2.34 work days each).



Early Start	Duration	Early Finish
Task Name		
Late Start	Slack	Late Finish



A completed network diagram created using [Microsoft Visio](#). Note the [critical path](#) is in red.

## See also

- [Project planning](#)
- [Beta distribution](#)
- [Triangular distribution](#)
- [Gantt chart](#)
- [Project network](#)
- [Project management](#)
- [Float \(project management\)](#)
- [Crash \(Schedule duration\)](#)

## External links

- [Open source software for making AON diagram](#)
- [The rudiments of PERT](#)
- [More explanation of PERT](#)
- [3 Point Estimating Tutorial on VisionaryTools.com](#)

Retrieved from "[http://en.wikipedia.org/wiki/Program\\_Evaluation\\_and\\_Review\\_Technique](http://en.wikipedia.org/wiki/Program_Evaluation_and_Review_Technique)"

# Critical path method

From Wikipedia, the free encyclopedia

(Redirected from [Critical path](#))

Jump to: [navigation](#), [search](#)

The **Critical Path Method**, abbreviated CPM, is a mathematically based algorithm for scheduling a set of project activities. It is a very important tool for effective [project management](#). It was developed in the 1950's in a joint venture between [DuPont Corporation](#) and [Remington Rand Corporation](#) for managing plant maintenance projects. Today, it is commonly used with all forms of projects, including construction, [software](#) development, research projects, product development, engineering, and plant maintenance, among others. Any project with interdependent activities can apply this method of scheduling.

The essential technique for using CPM is to construct a model of the project that includes the following:

1. A list of all activities required to complete the project (also known as [Work breakdown structure](#)),
2. The time (duration) that each activity will take to completion, and
3. The [dependencies](#) between the activities.

Using these values, CPM calculates the starting and ending times for each activity, determines which activities are critical to the completion of a project (called the **critical path**), and reveals those activities with "float time" (are less critical). In [project management](#), a **critical path** is the sequence of [project network](#) activities with the longest overall [duration](#), determining the shortest time possible to complete the project. Any delay of an activity on the critical path directly impacts the planned project completion date (i.e. there is no [float](#) on the critical path). A project can have several, parallel critical paths. An additional parallel path through the network with the total durations shorter than the critical path is called a sub-critical or non-critical path.

These results allow managers to prioritize activities for the effective management of project completion. Originally, the critical path method considered only logical [dependencies](#) among terminal elements. Since then, it has been expanded to allow for the inclusion of resources related to each activity. This capability allows for the exploration of a related concept called the [critical chain](#), which determines project duration based upon both time and resource dependencies.

Since project schedules change on a regular basis, CPM allows continuous monitoring of the schedule, allows the project manager to track the critical activities, and ensures that non-critical activities do not interfere with the critical ones. In addition, the method can easily incorporate the concepts of stochastic predictions, using the [Program Evaluation and Review Technique](#) (PERT).

Currently, there are several software solutions available in industry today that use the CPM method of scheduling, see [list of project management software](#). However, the method was developed and used (for decades) without the aid of computers (with pencil and paper).

However, there are drawbacks of this technique, as estimations are used to calculate times, if one mistake is made the whole analysis could be flawed causing major upset in the organisation of a project.

## See also

- [List of project management topics](#)
- [Program Evaluation and Review Technique](#)
- [Project](#)
- [Project management](#)
- [Project planning](#)
- [Work breakdown structure](#)
- [List of project management software](#)

ssd

## External links

- [How to use critical path analysis](#)
- [Open source Software for path analysis](#)
- [Critical Path scenarios in PmPedia](#)

Retrieved from "[http://en.wikipedia.org/wiki/Critical\\_path\\_method](http://en.wikipedia.org/wiki/Critical_path_method)"

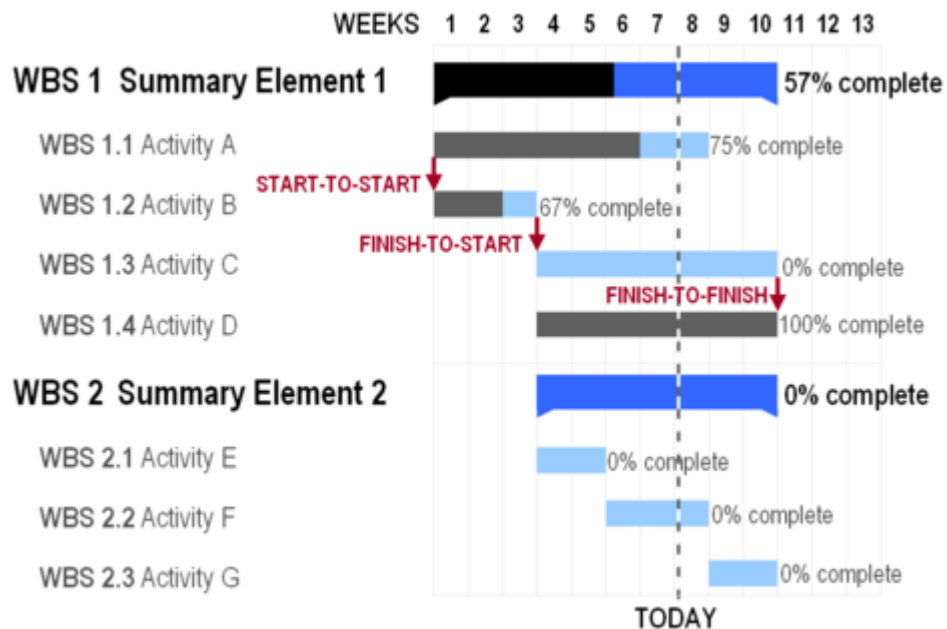
# Gantt chart

From Wikipedia, the free encyclopedia

(Redirected from [Gantt](#))

Jump to: [navigation](#), [search](#)

*"Gantt" redirects here. For other uses, see [Gantt \(disambiguation\)](#).*



Gantt chart showing three kinds of schedule dependencies (in red) and percent complete indications.

A **Gantt chart** is a popular type of [bar chart](#) that illustrates a [project schedule](#). Gantt charts illustrate the start and finish dates of the [terminal elements](#) and [summary elements](#) of a [project](#). Terminal elements and summary elements comprise the [work breakdown structure](#) of the project. Some Gantt charts also show the [dependency](#) (i.e., precedence network) relationships between activities. Gantt charts can be used to show current schedule status using percent-complete shadings and a vertical "TODAY" line (also called "TIME NOW"), as shown here.

## Historical development

The initial format of the chart was developed by [Henry Gantt](#) (1861-1919) in [1910](#) (see "Work, Wages and Profit" by H. L. Gantt, published by [The Engineering Magazine](#), [NY](#), 1910).

In the 1980s, personal computers eased the creation and editing of elaborate Gantt charts. These desktop applications were primarily intended for project managers and project schedulers. In the late 1990s and early 2000s, Gantt charts became a common feature of web-based applications, including collaborative [groupware](#).

Although now considered a common charting technique, Gantt charts were considered quite revolutionary at the time they were introduced. In recognition of Henry Gantt's contributions, the [Henry Laurence Gantt Medal](#) is awarded for distinguished achievement in management and service to the community.

## Advantages and limitations

Gantt charts have become a common technique for representing the phases and activities of a project [work breakdown structure](#), so they can be understood by a wide audience.

A common error made by those who equate Gantt chart design with project design is that they attempt to define the project [work breakdown structure](#) at the same time that they define schedule activities. This practice makes it very difficult to follow [100% Rule](#). Instead the WBS should be fully defined to follow 100% Rule, then the project schedule can be designed.

Although a Gantt chart is easily comprehended for small projects that fit on a single sheet or screen, they can become quite unwieldy for projects with more than about 30 activities. Larger Gantt charts may not be suitable for most computer displays. A related criticism is that Gantt charts communicate relatively little information per unit area of display. That is, projects are often considerably more complex than can be communicated effectively with a Gantt chart.

Gantt charts only represent part of the [triple constraints](#) of projects, because they focus primarily on schedule management. Moreover, Gantt charts do not represent the size of a project, therefore the magnitude of a behind-schedule condition is easily miscommunicated. If two projects are the same number of days behind schedule, the larger project has a larger impact on resource utilization, yet the Gantt does not represent this difference.

Although project management software can show schedule dependencies as lines between activities, displaying a large number of dependencies may result in a cluttered or unreadable chart.

Because the horizontal bars of a Gantt chart have a fixed height, they can misrepresent the planned workload (resource requirements) of a project. In the example shown in this article, Activities E and G appear to be the same size, but in reality they may be orders of magnitude different. A related criticism is that all activities of a Gantt chart show planned workload as constant. In practice, many activities (especially summary elements) have front-loaded or back-loaded work plans, so a Gantt chart with percent-complete shading may actually miscommunicate the true schedule performance status.

## External links

- [Long-running discussion](#) regarding limitations of the Gantt chart format, and alternatives, on [Edward Tufte's website](#)
- [Project schedule](#) from [University of Washington's project management site](#)
- [Project schedule](#)

Retrieved from "[http://en.wikipedia.org/wiki/Gantt\\_chart](http://en.wikipedia.org/wiki/Gantt_chart)"

# Work breakdown structure

From Wikipedia, the free encyclopedia

Jump to: [navigation](#), [search](#)

A **Work Breakdown Structure (WBS)** is a fundamental [project management](#) technique for defining and organizing the total [scope](#) of a [project](#), using a hierarchical [tree structure](#). The first two levels of the WBS (the root node and Level 2) define a set of *planned outcomes* that collectively and exclusively represent 100% of the project scope. At each subsequent level, the children of a parent node collectively and exclusively represent 100% of the scope of their parent node. A well-designed WBS describes planned outcomes instead of planned actions. Outcomes are the desired ends of the project, and can be predicted accurately; actions comprise the project plan and may be difficult to predict accurately. A well-designed WBS makes it easy to assign any project activity to one and only one [terminal element](#) of the WBS.

## Contents

- [1 WBS design principles](#)
  - [1.1 The 100% Rule](#)
  - [1.2 Planned outcomes, not planned actions](#)
  - [1.3 Mutually-exclusive elements](#)
  - [1.4 Level of detail \(granularity\) and progressive elaboration](#)
  - [1.5 WBS coding scheme](#)
- [2 WBS construction example](#)
- [3 Common pitfalls and misconceptions](#)
- [4 See also](#)
- [5 References](#)

## WBS design principles

### The 100% Rule

One of the most important WBS design principles is called the 100% Rule. The *Practice Standard for Work Breakdown Structures (Second Edition)*, published by the [Project Management Institute](#) (PMI) defines the 100% Rule as follows:

*The 100% Rule...states that the WBS includes 100% of the work defined by the project scope and captures ALL deliverables – internal, external, interim – in terms of the work to be completed, including project management. The 100% rule is one of the most important principles guiding the development, decomposition and evaluation of the WBS. The rule applies at all levels within the hierarchy: the sum of the work at the “child” level must equal 100% of the work represented by the “parent” and the WBS should not include any work that falls outside the actual scope of the project, that is, it cannot include more than 100% of the work... It is important to remember that the 100% rule also applies to the activity level. The work represented by the activities in each work package must add up to 100% of the work necessary to complete the work package. (p. 8)*

### Planned outcomes, not planned actions

If the WBS designer attempts to capture any action-oriented details in the WBS, he/she will likely include either too many actions or too few actions. Too many actions will exceed 100% of the parent's scope and too few will fall short of 100% of the parent's scope. The best way to adhere to the 100% Rule is to define WBS elements in terms of outcomes or results. This also ensures that the WBS is not overly prescriptive of methods, allowing for greater ingenuity and creative thinking on the part of the project participants. For new product development projects, the most common technique to assure an outcome-oriented WBS is to use a [product breakdown structure](#). [Feature-driven software projects](#) may use a similar technique which is to employ a feature breakdown structure. When a project provides professional services, a common technique is to capture all planned deliverables to create a deliverable-oriented WBS. Work breakdown structures that subdivide work by project phases (e.g. Preliminary Design Phase, Critical Design Phase) must ensure that phases are clearly separated by a deliverable (e.g. an approved Preliminary Design Review document, or an approved Critical Design Review document).

### Mutually-exclusive elements

In addition to the 100% Rule, it is important that there is no overlap in scope definition between two elements of a WBS. This ambiguity could result in duplicated work or miscommunications about responsibility and authority. Likewise, such overlap is likely to cause



confusion regarding project cost accounting. If the WBS element names are ambiguous, a WBS dictionary can help clarify the distinctions between WBS elements

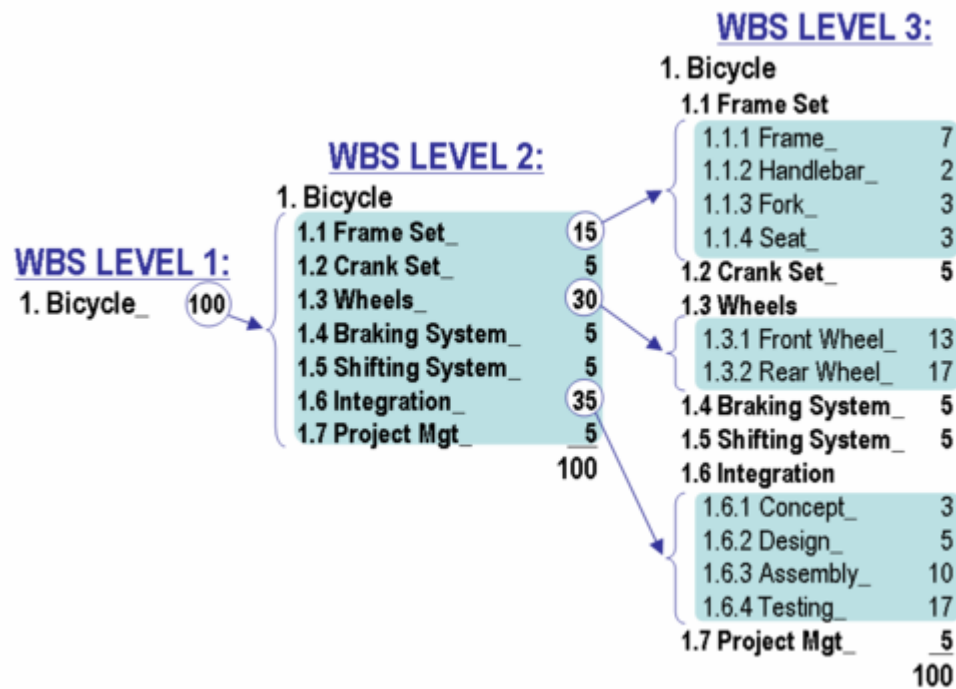
### **Level of detail (granularity) and progressive elaboration**

A question to be answered in the design of any WBS is when to stop dividing work into smaller elements. If a WBS terminal elements are defined too broadly, it may not be possible to track project performance effectively. If a WBS terminal elements are too granular, it may be inefficient to keep track of so many terminal elements, especially if the planned work is in the distant future. A satisfactory tradeoff may be found in the concept of *progressive elaboration* which allows WBS details to be progressively refined before work begins on an element of work. One form of progressive elaboration in large projects is called *rolling wave planning* which establishes a regular time schedule for progressive elaboration. In reality, an effective limit of WBS granularity may be reached when it is no longer possible to define planned outcomes, and the only details remaining are actions. Unless these actions can be defined to adhere to the 100% Rule, the WBS should not be further subdivided.

### **WBS coding scheme**

It is common for WBS elements to be numbered sequentially to reveal the hierarchical structure. For example **1.3.2 Front Wheel** identifies this item as a Level 3 WBS element, since there are three numbers separated [decimal point](#). A coding scheme also helps WBS elements to be recognized in any written context.

### **WBS construction example**



**Figure 1: WBS Construction Technique.** This exemplary WBS is from PMI's *Practice Standard for Work Breakdown Structures (2nd Edition)*. This image illustrates an objective method of employing the 100% Rule during WBS construction.

**Figure 1** shows a WBS construction technique that demonstrates the 100% Rule quantitatively. At the beginning of the design process, the project manager has assigned 100 points to the total scope of this project, which is designing and building a custom bicycle. At WBS Level 2, the 100 total points are subdivided into seven comprehensive elements. The number of points allocated to each is a judgment based on the relative effort involved; it is NOT an estimate of duration. The three largest elements of WBS Level 2 are further subdivided at Level 3, and so forth. The largest terminal elements at Level 3 represent only 17% of the total scope of work. These larger elements may be further subdivided using the *progressive elaboration* technique described above. In this example, the WBS coding scheme includes a trailing "underscore" character ("\_") to identify terminal elements. This is a useful coding scheme because planned activities (e.g. "Install inner tube and tire") will be assigned to terminal elements instead of parent elements. Incidentally, this quantitative method is related to the [Earned Value Management](#) technique.

It is recommended that WBS design be initiated with interactive software (e.g. a [spreadsheet](#)) that allows automatic rolling up of point values. Another recommended practice is to discuss the point estimations with project team members. This collaborative technique builds

greater insight into scope definitions, underlying assumptions, and consensus regarding the level of granularity required to manage the project.

## Common pitfalls and misconceptions

A WBS is not an exhaustive list of work. It is instead a comprehensive classification of project scope.

A WBS is not a project plan or a project schedule and it is not a chronological listing. It is considered poor practice to construct a project schedule (e.g. using [project management software](#)) before designing a proper WBS. This would be similar to scheduling the activities of home construction before completing the house design. Without concentrating on planned outcomes, it is very difficult to follow the 100% Rule at all levels of the WBS hierarchy. It is not possible to recover from an improperly defined WBS without starting over, so it is worthwhile to finish the WBS design before starting a project plan or project schedule.

A WBS is not an organizational hierarchy. Some practitioners make the mistake of creating a WBS that shadows the organizational chart. While it is common for responsibility to be *assigned* to organizational elements, a WBS that shadows the organizational structure is not descriptive of the project scope and is not outcome-oriented. See also: [responsibility assignment matrix](#).

Short-term memory capacity should not dictate the size and span of a WBS tree structure. Some reference material (including early versions of this Wikipedia article) suggest that each WBS level be limited to 5-9 elements because that is a theoretical limit to short-term memory. Such advice is part of the urban legend regarding [The Magical Number Seven, Plus or Minus Two](#), since WBS elements are certainly not random unconnected data. Definitive references regarding WBS construction do not contain such advice. It is far more important to construct a logical grouping of planned outcomes than to worry about the limits of short-term human memory.

WBS updates, other than progressive elaboration of details, require formal change control. This is another reason why a WBS should be outcome-oriented and not be prescriptive of methods. Methods can and do change frequently, but changes in planned outcomes require a higher degree of formality. If outcomes and actions are blended, change control may be too rigid for actions and too informal for outcomes.

## See also

- [list of project management topics](#)
- [project planning](#)
- [product breakdown structure](#)
- [project management software](#)

## References

- Carl L. Pritchard. *Nuts and Bolts Series 1: How to Build a Work Breakdown Structure*. [ISBN 1-890367-12-5](#)
- Project Management Institute. *Project Management Institute Practice Standard for Work Breakdown Structures, Second Edition (2006)*. [ISBN 1-933890-13-4](#) (Note: The Second Edition is an extensive re-write of the Practice Standard.)
- Gregory T. Haugan. *Effective Work Breakdown Structures (The Project Management Essential Library Series)*. [ISBN 1-56726-135-3](#)
- Dennis P. Miller, *Visual Project Planning & Scheduling, Second Edition (2002)*. [ISBN 0-9640630-2-6](#) (Note: This ebook is essential a facilitator's guide for planning a project based on the WBS.) [\[1\]](#)

Retrieved from "[http://en.wikipedia.org/wiki/Work\\_breakdown\\_structure](http://en.wikipedia.org/wiki/Work_breakdown_structure)"

## **Software Estimating Models: Three Viewpoints**

Dr. Randall W. Jensen , Software Technology Support Center  
Lawrence H. Putnam Sr. , Quantitative Software Management, Inc.  
William Roetzheim , Cost Xpert Group, Inc.

This article compares the approaches taken by three widely used models for software cost and schedule estimation. Each of the models is compared to a common framework of first-, second-, and third-order models to maintain consistency in the comparisons. The comparisons illustrate significant differences between the models, and show significant differences in the approaches used by each of the model classes.

The purpose of this article is to present an unbiased comparison of three approaches to estimating software development costs. Rather than a single author attempting to arrive at a middle of the road, politically correct description of the three approaches, this article presents the comparison according to three individuals who are at the heart of these major estimating philosophies (from the horses' mouths so to speak).

### **Origins and Evolution of Software Models**

This article prompted an enlightening trip back into the fuzzy history of computer-based software cost and schedule estimating methods and tools. It appears that the origins are not that remote, and the methods appeared over a relatively short period of time and have evolved in spurts and starts since then. The first real contribution to the estimating technology happened in 1958 with the introduction of the Norden staffing profile [1]. This profile has been incorporated in many of the estimating methodologies introduced since then.

A flurry of software estimating methods was introduced beginning in the mid-1970s and throughout the next decade. The first publication of any significance was presented by Ray Wolverton [2] of TRW in 1974. Wolverton was a major contributor to the development of the Constructive Cost Model (COCOMO) [3]. The second major contribution to the evolution of software estimating tools was the Doty Associates model [4], developed for the U.S. Air Force in 1977. The period from 1974 through 1981 brought most of the software estimating models (tools) we use today to the marketplace.

Each of these tools evolved at a gentle pace (refined algorithms and drivers) until about 1995, when significant changes were made to many of the models. COCOMO II, for example, had several releases between 1995 and 1999. Sage, released in 1995, is a major redefinition of the 1979 Seer model. Cost Xpert, introduced in 1996, is a major modification of the COCOMO family line. It is amazing

that, in the 25 years elapsed since the first wave of estimating tools, development environments have changed so little that these models and their predicted environments are still valid today.

## **Framework for Discussion**

When we look at software estimating models, they generally fall into one of three classes: first-, second- or third-order forms. This article compares three widely used models using the three classes as a framework for discussion and comparison.

### **First-Order Model**

The first-order model is the most rudimentary model class. The model is simply a productivity constant, defining the production capability of the development organization in terms of arbitrary production units multiplied by the software product effective size to obtain the development effort or cost. The production units can be source lines of code (SLOC), function points (FPs), object points, use cases, and a host of other units. For the purpose of this discussion, we will use effective source lines of code (ESLOC) as the production measure, and person-hours per ESLOC as the productivity measure. This can be stated as follows:

$$E_d = C_k S_e \quad (1)$$

where

**$E_d$  is the development effort in person hours (ph).**

**$C_k$  is a productivity factor (ph/esloc).**

**$S_e$  is the number of ESLOC.**

The productivity factor is commonly determined by the product type, historic developer capability, or both as derived from past projects. As simple as this equation is, it is widely used to produce high-level, rough estimates. An expansion of this form used as far back as the 1970s is:

$$E_d = C_k(S_{\text{new}} + 0.75S_{\text{modified}} + 0.2S_{\text{reused}}) \text{ ph} \quad (2)$$

Or it is a similar variation. The weakness of this model is its insensitivity to the magnitude of the effective product size. Productivity is, or at least should be, decreased for larger projects.

## Second-Order Model

The second-order model compensates for the productivity decrease in larger projects by incorporating an *entropy* factor to account for the productivity change. The entropy effect demonstrates the impact of a large number of communications paths that are present in large development teams. The number of paths is specified by  $n(n-1)/2$  where  $n$  is the number of development personnel. The second-order model becomes the following:

$$E_s = C_1 S_e^\beta \quad (3)$$

where

**$\beta$  is an entropy factor that accounts for the productivity change as a function of effective product size.**

An entropy factor value of 1.0 represents no productivity change with size. An entropy value of less than 1.0 shows a productivity increase with size, and a value greater than 1.0 represents a productivity decrease with size. Entropy values of less than 1.0 are inconsistent with historical software data. Most of the widely used models in the 1980s (COCOMO embedded mode, PRICE-S, REVIC, Seer, and SLIM) used entropy values of approximately 1.2 for Department of Defense projects.

The major weakness of this model is its inability to adjust the productivity factor to account for variations between projects in development environments. For example, contractor A may have a more efficient process than contractor B; however, contractor A may be using a development team with less experience than used in the historic productivity factor. Different constraints may be present in the current development than was present in previous projects. In addition, using a fixed, or calibrated, productivity factor limits the model's application across a wide variety of environments.

## Third-Order Model

The third-order model compensates for the second-order model's narrow applicability by incorporating a set of environment factors to adjust the productivity factor to fit a larger range of problems. The form of this model is as follows:

$$E_s = C_2 \left( \prod_{i=1}^n f_i \right) S_e^\beta \quad (4)$$

where

**$f_i$  is the  $i$ th environment factor.**

**$n$  is the number of environment factors.**

The number of environment factors varies across estimating models, and is typically between 15 and 32.

Using the generalized model set defined in equations (1) through (4), we have a framework for comparing the definition and features of the software estimating models. Three model types are described and compared in the following sections of this article: (1) models evolving from the 1979 Seer model developed and described by Dr. Randall Jensen, (2) models evolving from the COCOMO model described by William Roetzheim, and (3) the SLIM model developed and described by Lawrence Putnam, Sr.

### **Sage/Seer Effort and Schedule Calculations**

Software development involves three important elements: people, a process, and a product. The people element describes management approach and style as well as personnel attributes, including capability, motivation, and communication effectiveness. Process represents the software development approach, tools, practices, and life-cycle definition. The product attributes include project-imposed constraints such as development standard, memory and time constraints, and security issues.

Software development is the most communication-intensive of all engineering processes. This unique software process characteristic suggests significant productivity gains are more likely to be realized through communication improvement, rather than through technology. Communication effectiveness is determined by organizational structure, management approach, and development environment. The Jensen model [5], and its implementations, embodies the impacts of the three important elements in software development cost and schedule estimates.

This section of the article discusses the underlying theory of a line of software cost and schedule estimating tools that evolved from the Jensen software model [6] at Hughes Aircraft Company's Space and Communications Group in 1979. The original model implementation was called Seer (not an acronym), a name later converted to the acronym SEER-SEM (Software Evaluation and Estimation of Resources-Software Estimating Model) [7] and trademarked by Galorath Associates, Inc. (GAI) in 1990. The Seer concepts were influenced by Lawrence Putnam's work [8] published in 1976 and the Doty Associates [9] estimating model published in 1977. The Seer model was derived from the U.S. Army data used by Putnam to develop SLIM, but with a different interpretation of the data itself.

The first major update to the Jensen model came in 1995 (Jensen II) with the addition of project management and personnel characteristics effects to the model. The impacts of motivation, management style, and teaming on productivity have long been suspected, but until 1995 the data to support the alleged behavior was simply insufficient to make credible model changes. These changes were implemented in the Software Engineering, Inc., Sage [10] software estimating system. For the sake of brevity, the Jensen model (I



and II) will be referred to as Sage throughout this discussion. The following discussion applies to all descendants of the original Jensen estimating model. The fundamental equations are the following:

$$S_s = C_{te} \sqrt{KT_d} \quad (5)$$

and

$$D = \frac{K}{T_d^3} \quad (6)$$

where

**$C_{te}$  is the effective technology constant of the development activity.**

**$K$  is the total life-cycle effort in person- years (py) of the software development starting with the software requirements review through the software's end of life.**

**$T_d$  is the software product development time in years.**

**$D$  is the product complexity rating.**

Equations (5) and (6) solved simultaneously provide both schedule and effort estimates in one calculation with the relationship between effort and schedule linked by the product complexity. This approach is unique to the Jensen and Putnam models.

The parameter  $D$  is the same as the *Difficulty* parameter discovered by Putnam. This explains the use of  $D$  to describe what the Jensen model refers to as complexity.

Development effort is defined by

$$E_d = 0.3945K$$

where

**$E_d$  is the development effort in py through the final qualification test.**

The Sage software development effort equation is an implementation of the third-order model discussed in the introduction to the model comparison even though it is not immediately apparent. Combining equations (5) and (6), we find the following:

$$E_s = \frac{18,797^{S_s}}{C_{te}^{S_s}} S_s^{12} \text{ person months (pm)} \quad (7)$$

The ugly part of equation (7) preceding the effective size element comprises the productivity factor of the third order model. The effective technology constant  $C_{te}$  contains two components: (1) the basic technology constant  $C_{tb}$  representing the development organization's raw capability; that is, capability independent of the constraints imposed by a specific development, and (2) the impacts of 24 environment factors/constraints. The  $C_{te}$  value is obtained from the following:

$$C_{te} = \frac{C_{tb}}{\prod_{i=1}^{24} f_i} \quad (8)$$

where

**$C_{tb}$  represents the basic technology constant.**

**$f_i$  is the  $i$ th product-impacted environment factor.**

The  $C_{tb}$  value can be anywhere between 2,000 and 20,000 with a normal range between 5,500 and 7,500. The highest value observed from available data at this time is about 8,635. Higher values obviously imply higher productivity and efficiency. Theoretical values of  $C_{te}$  range from 0 through 20,000. The practical upper  $C_{te}$  limit is defined by an organization's rating. The practical lower  $C_{te}$  bound is about 500 for a less-than-average organization and severe product constraints.

The relative cost impact of the analyst capability rating for Sage is shown in Table 1 as an example of one of the 24 environment factors.

Definition	Highly motivated AND experienced team organization	Highly motivated OR experienced team organization	Traditional software development organization	Poorly motivated OR non-associative organization	Poorly motivated AND non-associative organization
Relative cost impact	0.71	0.86	1.00	1.19	1.46

Table 1: Sage Analyst Capability Ratings

The product development time  $T_d$  is the minimum development time as can be seen from the Paul Masson cost-time relationship shown in Figure 1. The *Paul Masson Point* represents the minimum development time that can be achieved with a specified size, environment, and complexity. By attempting to reduce the schedule below the minimum time, the cost will increase and the schedule will also increase as described by Brooks Law [11]: “Adding people to a late software project makes it later.” Sage computes the minimum development schedule as a default.

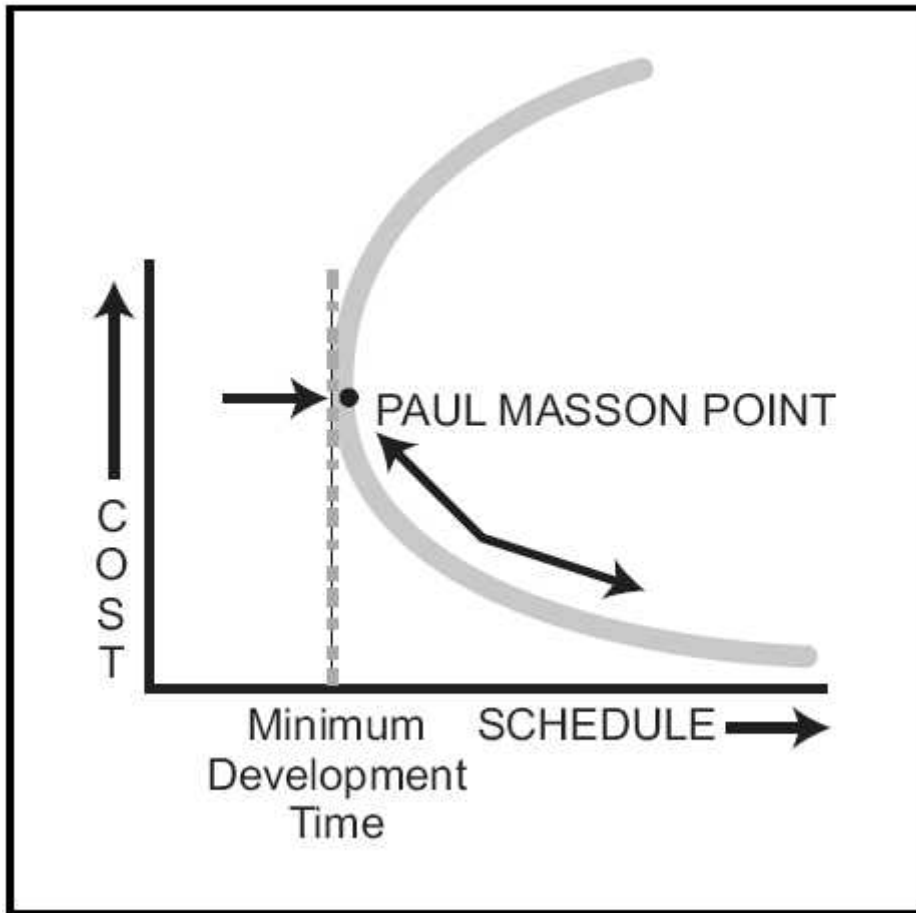


Figure 1: Paul Masson Rule

The region shown by the double-headed arrow is represented as a square-law relationship in the model between cost and schedule, or  $c = KT^2_d$ . At first glance it seems that a longer schedule should equate to higher cost. By explaining the phenomenon in two logical steps, the productivity gain over the region becomes clear. A longer schedule requires a smaller development team. A smaller team is more efficient; thus, productivity improves and the cost decreases. This phenomenon applies until the productivity gain is eaten up by fixed costs.

As an example of the effort and schedule predicted by Sage, let us assume the following development project parameters: The product is a satellite mission operations system (application with significant logical complexity with some changes to the underlying operating system) consisting of 59,400 new SLOC. A basic technology rating of 7,606 places the developer at the upper end of the typical

capability range. The project environment constraints, including the Institute of Electrical and Electronics Engineers Standard 12207 development standard, reduce the effective technology rating to 2,624. The third-order equation form of equation (7) reduces to  $E_d = 4.007S^{1.2}_e$ . The results are tabulated in Table 2.

$S_e$ kesloc	$D$	$C_{tb}$	$C_{te}$	$E_d$ , pm	$T_d$ mo	Productivity, sloc/pm
59.4	12	7,606	2,624	538.7	25	110

Table 2: Example Estimate

The minimum development schedule, which is simultaneously calculated in equation (5), is approximately 25 months.

## Quantitative Software Management View of Software Estimating and Productivity Measurement

Productivity measurement is used to:

1. Tune estimating systems.
2. Baseline and measure progress in software development.

But what is *productivity*? My answer is: It is not SLOC/PM or FP/PM. This is the traditional view from economic theory – output/input.

The software industry has 35 years of experience that shows that this ratio works poorly as a productivity metric. At Quantitative Software Management (QSM), we have learned why it does not work: because it ignores schedule, and software development is very sensitive to schedule. A vast amount of our 27-year collection of data, some 6,600 completed systems, coupled with empirical analysis shows that schedule is the most important factor in estimating relationships and must be dealt with explicitly. If schedule is not so recognized and dealt with, then it will assert itself implicitly and cause much grief. By this, I mean both time and effort must be included multiplicatively in an expression for a good software algorithm. We have found this to be of the conceptual form:

$$\text{Amount of function} = \text{Effort} * \text{Schedule} * \text{Process Productivity (9)}$$

where

**Effort and Schedule have exponents. Specifically,**

$$\text{Size} = (\text{Effort}/\beta)^{1/3} \text{Schedule}^{4/3} \text{Process Productivity Parameter (10)}$$

where

*Size* is the size in SLOC, or other measure of amount of function.

*Effort* is the amount of development effort in py.

*beta* is a special skills factor that varies as a function of size from 0.16 to 0.39.

*beta* has the effect of reducing process productivity, for estimating purposes, as the need for integration, testing, quality assurance, documentation, and management skills grows with increased complexity resulting from the increase in size.

*Schedule* is the development time in years.

*Process Productivity Parameter* is the productivity number that we use to tune the model to the capability of the organization and the difficulty of the application. We do this by calibration as explained in the next section. The theoretical range of values is from 610 to 1,346,269. The range of values seen in practice across all application types is 1,974 to 121,393 and varies exponentially.

## Estimating and Tuning Models

All models need tuning to moderate the *productivity* adjusting factor.

- Some models use effort multipliers or modifiers.
- I have found that we can use the software equation to *calibrate* our estimating algorithm. This calibration process is far more accurate than intelligent guesses of the settings for effort multipliers because it is based on real data from the development organization. All we need to do is to rearrange the software equation into this form:

$$\text{Process Productivity Parameter} = \text{Size} / ((\text{Effort}/\beta)^{1/3} (\text{Schedule}^{4/3})) \text{ (11)}$$

From historic projects we know the size (SLOC, FPs, etc.), effort (py) and schedule (years). Then just put in a consistent set of historic numbers and we can calculate a Process Productivity Parameter. This works well. Note that the expression for Process Productivity Parameter includes schedule and that it is multiplicatively tied to effort. This expression is our definition of software productivity.

This software equation has two variables that we want to solve for: schedule and effort. That means we have to have another equation to get a solution in the form of a schedule-effort pair. The second equation may be in the form of a constraint like maximum budget for the project (Maximum Development Effort = Maximum Cost/\$Average Burdened Labor Rate), or, Maximum Schedule = Maximum Development Time in years. There are a number of other constraints that can be used such as peak manpower, or maximum manpower buildup rate (defined as *Difficulty* in Randall Jensen's preceding section).

### Example of an Estimate

Here is a simple example: We need an estimate for a Global Positioning System navigation system for an air-launched, land-attack missile. We have completed the high-level design phase. Estimated size is 40,000 C++ SLOC; Process Productivity Parameter for this class of work (real time avionic system) is 3,194 [taken from Table 14.8, in 12],  $\beta = 0.34$  [taken from Table 14.4, in 13]. We have to deliver the system to the customer for operational service in two years (24 months from the end of high-level design) The fully burdened contractor labor rate is \$200,000 per person-year. Substituting in the software equation and solving for effort, we have the following:

$$40,000 = (\text{Effort}/0.34)^{1/3} 2^{4/3} 3,194$$

**Effort = 41.74 PY (Approximately 500.9 pm)**  
**Cost = \$200,000/ PY \* 41.74 PY = \$8.35 million**

### Observations Concerning Effort Multipliers/Moderators

Many estimating systems use a family of adjusting factors to try to tune their productivity constant. Between 15 and 25 of these *tweakers* are typical. The values are picked from a scale centered on 1.0 that increase or decrease the Productivity constant. This process does moderate the baseline productivity value. Unfortunately, it is highly subjective – dependent upon the judgment of the practitioner. It is not consistent or reproducible from person to person and hence it introduces considerable uncertainty into the estimate that follows.

At QSM, we use a family of tweakers for tools and methods, technical complexity of the project, competence, experience, and skill of the development team. This list is similar to most other estimating systems. But we use it only as a *secondary* technique for those organizations that truly have no historic data. The notion of calibration from historic data is much more accurate and powerful because it captures the real capability and character of the development organization in a single number – the Process Productivity Parameter in the software equation. This single number is unambiguous and consistent; there is no subjectivity involved.

### Benchmarking

One of the nice things about being able to substitute your historical data into the software equation is that you can calculate an unambiguous number that can be used for benchmarking. We transform the Process Productivity into a linear scale and call that a

Productivity Index (PI). If we collect a homogeneous set of data from a development organization and calculate the PI for each project, we see a fairly normal distribution with a central tendency.

The central tendency represents our current average PI. If we keep track of all our projects over time and if we are doing process improvement (trying to move up the SEI scale) then we will see an increase in the PI over time. Often we can plot the PI behavior over time and pick up the trend. Moreover, this graphical approach makes it easy to compare organizations doing similar types of work. Extending this thinking a little bit provides the ability to quantitatively compare the real capability of bidders on a software development contract. This comparison process takes a lot of guesswork out of trying to determine the real capability of vendors.

The benchmarking idea can be extended easily to show performance capabilities of the development organization. The idea is to take a fairly large body of contemporaneous historic data from the same industry sector, then generate some trend lines plots of the generic form: management metric (schedule, effort, staffing, defects, etc.) versus size (SLOC, FPs, objects, etc.). Next, superimpose data points from the development organization being measured on top of these trend lines and see how they position (high or low) compared with the industry average trend line at the appropriate size. For example, if your data shows a pattern of falling below the average line for effort, schedule, and defects you are a more effective producer (high productivity developer). Almost invariably your PI and Mean Time to Defect (MTTD) will be higher as well. This means you can unambiguously and quantitatively measure productivity. After nearly 30 years of experience doing it, we know it works consistently and well. An example of such plots is shown in Figure 2.



## Developer Completed Projects Compared to QSM Command and Control System Trend Lines

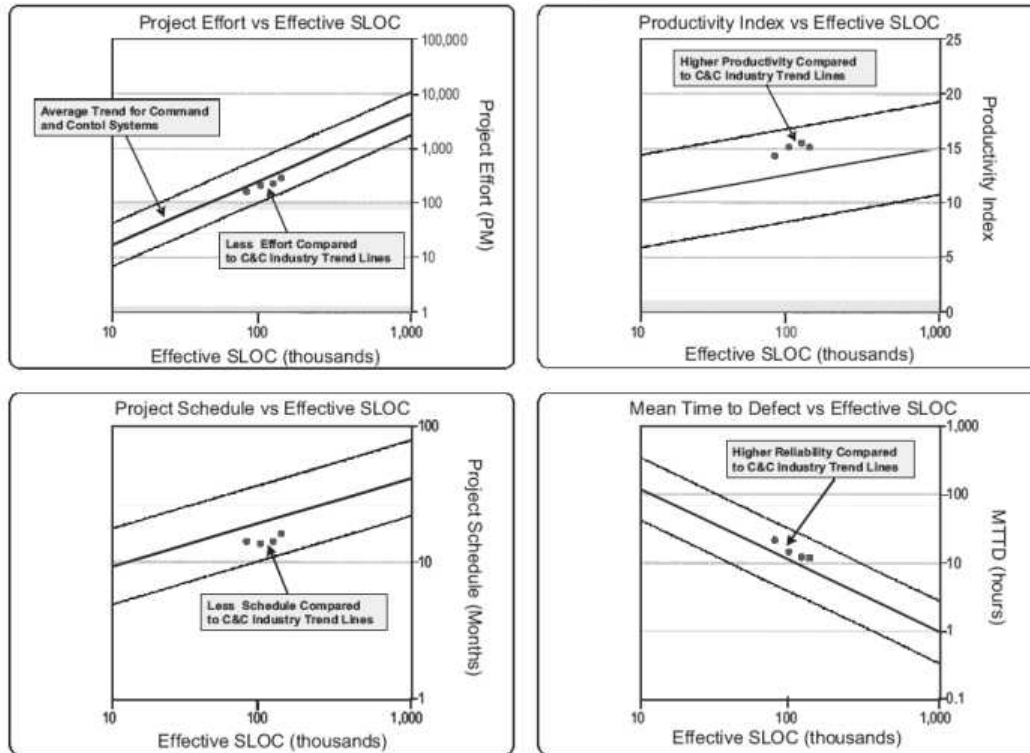


Figure 2: Trend Lines Plots

## Cost Xpert Effort and Schedule Calculations

Algorithmically, Cost Xpert started with the COCOMO models, added Revised Intermediate COCOMO (REVIC) extensions, and then layered functionality on top of this base. The core approach of Cost Xpert is as follows:

1. Define scope using various measures of size (e.g., SLOC, FPs, class-method points, user stories, many others) for new development, reused code, and commercial off-the-shelf (COTS) components).
2. Use scope and a variety of adjusting factors (addressed below) to calculate effort.
3. Use effort to calculate optimal schedule.
4. Feedback deviations from the optimal schedule to adjust effort, if necessary.

## First-Order Modeling in Cost Xpert

First order modeling involves linear calculations of effort using a productivity constant:

$$\text{Effort} = \text{Productivity} * \text{Size (12)}$$

In Cost Xpert, all sizing metrics (SLOC, FPs, class-method points, user stories, etc.) and development stage (new, reused, COTS) are normalized to a SLOC equivalent. This SLOC equivalent is defined such that it is valid for estimating effort, although in some situations it may not accurately represent physical lines of code (for example, in environments where much of the code is auto-generated). In other words, although it once represented physical lines of code calculated using an approach called backfiring, and although that relationship is still true for older development environments, with newer environments it has become more of an estimating size proxy.

COCOMO and REVIC use(d) relatively small databases of projects and have a correspondingly small universe of productivity numbers. For example, COCOMO II uses the value 2.94 person-months per thousand SLOC (KSLOC) [14]. Commercial vendors, including the Cost Xpert Group, are able to maintain much larger databases of projects and hence can segment those databases into a finer granularity of project classes, each with a corresponding productivity number.

Cost Xpert uses a project type variable to denote the nature of the project (e.g., military, commercial, and internet) and set the coefficient for given historic database segments. Actual numbers for these sample project classes are shown in Table 3 [15]. The productivity number multiplied by KSLOC yields the first order effort in person-months.

Project Type	Productivity Factor (pm/KSLOC)
Military	3.97
Commercial	2.40
Internet	2.51

Table 3: Productivity Factors

However, Cost Xpert Group research has determined that there are additional sources of variation across projects and project domains (see the Third-Order Modeling section for examples). Our calibrations account for the additional sources of variation and therefore produce different coefficients for given project classes than the models with reduced factor sets. The overall net productivity in Cost Xpert thus accounts for second-order and third-order modeling described in the following sections.

## Second-Order Modeling in Cost Xpert

Second-order modeling in Cost Xpert involves adjusting the productivity to allow for economies or diseconomies of scale. An economy of scale indicates that the productivity goes up as the scope goes up. For example, you would expect that it is cheaper per yard to install 100,000 yards of carpeting than 1,000 yards. A diseconomy of scale indicates that the productivity goes down as the scope goes up. In software, we are dealing with diseconomies (larger projects are less efficient). This is modeled by raising the size to a power, with a power greater than 1.0 increasing the apparent scope, and thereby effort, with increasing project size. The second-order model looks like this:

$$\text{Effort} = \text{Productivity Factor} * \text{Size}^{\text{Scaling Factor}} \quad (13)$$

Table 4 shows some scaling factors by project type.

Project Type	Scaling Factor
Military, Complex	1.197
Military, Average	1.120
Military, Simple	1.054
Commercial	1.054

Table 4: Scaling Factors for Various Project Types

## Third-Order Modeling in Cost Xpert

Third-order modeling in Cost Xpert involves adjusting the productivity and scaling factors by project-specific variables. Cost Xpert uses 32 variables, called environment variables (E) to adjust the productivity number up or down, and five variables, called scaling variables (S) to adjust the scaling factor up or down. Each variable is set to a value ranging from very low to extremely high using defined criteria for each setting. Many of these variables will typically be fixed at a given value for an organization, with only a handful actually varying from project to project. The total productivity factor adjustment (PFA) is the product of the individual productivity factor values.

$$\text{PFA} = \prod E \quad (14)$$

Table 5 shows some sample environmental variables and the resultant PFAs as a sample of how this works.

	Required Reliability (E)		Multi-Site Development (E)		Security Classification (E)		Net Productivity (PFA)
Military	Very High	1.26	Nominal	1.00	High	1.10	1.386
Commercial	Nominal	1.00	Very High	0.86	Nominal	1.00	0.86

Table 5: Sample Productivity Factor Adjustments

The higher the number, the more effort required to deliver the same KSLOC, so in the Table 5 sample the commercial product would require less effort per KSLOC than the military project.

Additionally, Cost Xpert has introduced different types of linear factors for important sources of effort variation. We find that much of the variance between project classes (e.g., military versus commercial) can be accounted for by the life cycles and standards typically employed. We have introduced the following factors:

- **Project Life Cycle:** The life cycle (template of activities, or work breakdown structure) used for development).
- **Project Standard:** The deliverables (engineering specifications, or artifacts) produced during development.

These factors are rated by selecting named life cycles and standards, which are different than rating the standard environmental variables. In addition to adjusting effort, the life cycle and standard are used to create specific detailed project plans and page size estimates. They are also used in our defect model. Examples using actual numbers for sample projects are shown in Table 6, where RAD is rapid application design. These two samples could represent military and commercial projects respectively.

	Life Cycle and Multiplier		Standard and Multiplier		Net Productivity
Sample 1	Waterfall	1.01	Military-498	1.34	1.35
Sample 2	RAD	0.91	RAD	0.51	0.46

Table 6: Life Cycle and Standard Adjustment Factors

The relative productivity difference between the samples due to these two factors would be  $1.35/.46 = 2.9$  or 290%.

The five scaling variables (S) (not shown) work in a somewhat analogous manner, but the five factors are summed to adjust the exponential factor that applies to the diseconomy of scale. The total third-order formula is then the following:

$$\text{Effort} = \prod E * P * KSLOC^{\text{Scaling Factor}(S)} \quad (15)$$

where

**Effort** is the effort in pm.

**E** are the various environmental factors.

**P** is the productivity factor (which is further broken down into a project type, life cycle, and standard).

**KSLOC** is the SLOC equivalent in thousands.

**S** are the five scaling factor adjustments.

**ScaleFactor** is the default scaling factor for this project type.

If we use the military productivity factor from Table 3, military-complex from Table 4, the military PF adjustment from Table 5, and the life cycle and standard adjustments for sample 1 in Table 6, the equation simplifies to:

$$\begin{aligned} \text{Effort} &= 3.97 * 1.386 * 1.35 * KSLOC^{(1.197 * (12.575))} \\ \text{Effort} &= 7.49 * KSLOC^{(1.197 * (12.575))} \end{aligned} \quad (16)$$

## Calculating Schedule and Adjusting for Schedule Deviations

In Cost Xpert, the optimal schedule is driven by the calculated effort. The schedule formula is of the form:

$$\text{Schedule} = \alpha * \text{Effort}^{\beta} \quad (17)$$

where

**Schedule** is the schedule in calendar months.

**α** is a linear constant.

**β** is an exponential constant.

Table 7 shows a couple of sample values.

Project Type	$\alpha$	$\beta$
Military	3.80	0.378
Commerical	2.50	0.3348

Table 7: Sample Schedule Factors

Accelerating a project from this calculated schedule results in inefficiencies that then lower productivity and increase effort. Cost Xpert handles this schedule acceleration adjustment to productivity through a table lookup and interpolation between points in the table.

## Summary and Conclusions

The three model implementations described in this article represent the majority of the estimating approaches available to the estimator today. Our intent in this article is not to advocate a single software estimating approach or tool, but is to expose you, the reader, to the mindsets incorporated in the more widely used approaches available today.

## References

1. Norden, P.V. "Curve Fitting for a Model of Applied Research and Development Scheduling." *IBM Journal of Research and Development* 3 (July 1958).
2. Wolverton, R.W. "The Cost of Developing Large-Scale Software." *IEEE Transactions on Computers* June 1974: 615-636.
3. Boehm, B.W. *Software Engineering Economics*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1981.
4. Herd, J.R., J.N. Postak, We. E. Russell, and K.R. Stewart. "Software Cost Estimation Study – Final Technical Report." Vol. I. RADC-TR-77-220. Rockville, MD: Doty Associates, Inc., June 1977.
5. Jensen, R.W. "Management Impact of Software Cost and Schedule." July, 1996:6.
6. Jensen, R.W. *A Macrolevel Software Development Cost Estimation Methodology*. Proc. of the Fourteenth Asilomar Conference on Circuits, Systems and Computers. Pacific Grove, CA, 17-19 Nov. 1980.
7. Galorath, Inc. *SEER-SEM Users Manual*. El Segundo, CA: Galorath Inc., Mar. 2001.
8. Putnam, L.H. *A Macro-Estimating Methodology for Software Development*. Proc. of IEEE COMPCON '76 Fall, Sept. 1976: 138-143.
9. Herd, J.R., J.N. Postak, We. E. Russell, and K.R. Stewart. "Software Cost Estimation Study – Final Technical Report." Vol. I. RADC-TR-77-220. Rockville, MD: Doty Associates, Inc., June 1977.
10. Software Engineering, Inc. *Sage User's Manual*. Brigham City, UT: Software Engineering, Inc., 1995.
11. Brooks Jr., F.P. *The Mythical Man-Month*. Reading, MA: Addison-Wesley, 1975.

12. Putnam, Lawrence H., and Ware Myers. *Measures for Excellence* . Englewood Cliffs, NJ: Prentice-Hall, Inc., 1992: 237
13. Putnam. *Measures for Excellence* , 234.
14. Boehm B., et al. *Software Cost Estimation With COCOMO II* . Prentice-Hall, 2000.
15. Cost Xpert Group, Inc. *Cost Xpert 3.3 Users Manual* . San Diego, CA: Cost Xpert Group, Inc., 2003.

## Note

1. This is a good place to point out a major difference between software and almost any other manufactured product. In other estimating areas, a large number of products improves productivity through the ability to spread costs over a large sample and reduce learning curve effects. The software product is but a single production item that becomes more complex to manage and develop as the effective size increases.

## About the Authors



**Randall W. Jensen, Ph.D.**, is a consultant for the Software Technology Support Center, Hill Air Force Base, with more than 40 years of practical experience as a computer professional in hardware and software development. He developed the model that underlies the Sage and the Galorath, Inc. SEER-SEM software cost and schedule estimating systems. He retired as chief scientist in the Software Engineering Division of Hughes Aircraft Company's Ground Systems Group. Jensen founded Software Engineering, Inc., a software management-consulting firm in 1980. Jensen received the International Society of Parametric Analysts Freiman Award for Outstanding Contributions to Parametric Estimating in 1984. He has published several computer-related texts, including "Software Engineering," and numerous software and hardware analysis papers. He has a Bachelor of Science, a Master of Science, and a doctorate all in electrical engineering from Utah State University.

Software Technology Support Center  
6022 Fir AVE BLDG 1238  
Hill AFB, UT 84056  
Phone: (801) 775-5742  
Fax: (801) 777-8069  
E-mail: [randall.jensen@hill.af.mil](mailto:randall.jensen@hill.af.mil)





**Lawrence H. Putnam Sr.** is the founder and chief executive officer of Quantitative Software Management, Inc., a developer of commercial software estimating, benchmarking, and control tools known under the trademark SLIM. He served 26 years on active duty in the U.S. Army and retired as a colonel. Putnam has been deeply involved in the quantitative aspects of software management for the past 30 years. He is a member of Sigma Xi, Association for Computing Machinery, Institute of Electrical and Electronics Engineers (IEEE), and IEEE Computer Society. He was presented the Freiman Award for outstanding work in parametric modeling by the International Society of Parametric Analysts. He is the co-author of five books on software estimating, control, and benchmarking. Putnam has a Bachelor of Science from the United States Military Academy and a Master of Science in physics from the Naval Postgraduate School.

Quantitative Software  
Management, Inc.  
2000 Corporate Ridge STE 900  
McLean, VA 22102  
Phone: (703) 790-0055  
Fax: (703) 749-3795  
E-mail: [larry\\_putnam\\_sr@qsm.com](mailto:larry_putnam_sr@qsm.com)



**William Roetzheim** is the founder of the Cost Xpert Group, Inc., a Jamul-based organization specializing in software cost estimation tools, training, processes, and consulting. He has 25 years experience in the software industry, is the author of 15 software related books, and over 100 technical articles.

2990 Jamacha RD STE 250  
Rancho San Diego, CA 92019  
Phone: (619) 917-4917  
Fax: (619) 374-7311  
E-mail: [william@costXpert.com](mailto:william@costXpert.com)