The 10 Step Software Estimation Process For Successful Software Planning, Measurement and Control

Daniel D. Galorath
Galorath Incorporated
www.galorath.com

Abstract

An effective software estimate provides the information needed to design a workable software development plan. How well the project is estimated is ultimately the key to the project's (and product's) success. An effective software estimate provides important information for making project decisions, projecting performance, and defining objectives and plans. Without the proper guidance in a project, the results could be disastrous. Viable estimation is extremely valuable for project success for nearly any software project from small agile projects to huge projects.

The focus of this paper is how to make software projects more successful by properly estimating and planning costs, schedules, risks, and resources. It begins by covering the fundamental problems of unreasonable software estimation: not planning up front; failure to use viable estimates as the bases of an achievable project plan, not updating the plan and estimates when a project changes, and failing to consider the uncertainties inherent in estimates. Most estimates are prepared early on in the life cycle of a project, when there are typically a large number of undefined areas related to the project. The steps presented in this paper provide a complete method for developing estimates and plans.

This paper proposes a 10-step estimation process that begins by addressing the need for project metrics and the fundamental software estimation concepts. It shows how to build a viable project estimate, which includes the work involved in the actual generation of an estimate, including sizing the software, generating the actual software project estimate, and performing risk/uncertainty analysis. Finally the process rounds out with a discussion on validation of the estimate, obtaining lessons learned, and use of the estimate throughout the project. Based on the book, Software Sizing, Estimation, and Risk Management: When Performance is Measured Performance Improves, by Daniel D. Galorath and Michael W. Evans (Auerbach Publications, February 2006, ISBN: 0849335930)

Introduction

Too many software projects fail. And more of these failures are due to planning inadequacies than unachievable technical objectives, failed technology, or unachievable requirements.

A software estimation process that is integrated with the software development process can help projects establish realistic and credible plans to implement the project requirements and satisfy commitments. It also can support other management activities by providing accurate and timely planning information.

Many elements are involved in determining the structure of a project, including requirements, architecture, quality provisions, and staffing mix. Perhaps the most important element in the success or failure of a project are estimates of its scope, in terms of both the time and cost that will be required and the plans based on those estimates. The estimate drives every aspect of the project, constrains the actions that can be taken in the development or upgrade of a product, and limits available options. Although many people think they can estimate project scope based on their engineering or management experience, most off-the-cuff estimates are incorrect and are most often based on simple assumptions and over-optimism, or worse, are made to accord with what others want to hear. Needless to say, such estimates often lead to disaster.

If the estimate is unrealistically low, the project will be understaffed from its outset and, worse still, the resulting excessive overtime or staff burnout will cause attrition and compound the problems facing the project. Overestimation is not the answer. Indeed, overestimating a project can have the same effects as any other inaccurate estimate.

The definition of the verb to estimate is to produce a statement of the approximate value of some quantity. Estimates are based upon incomplete, imperfect knowledge and assumptions about the future. Most importantly, however, all estimates have uncertainty.

Ideally an estimate should be produced using the ten-step process described in Figure 1.

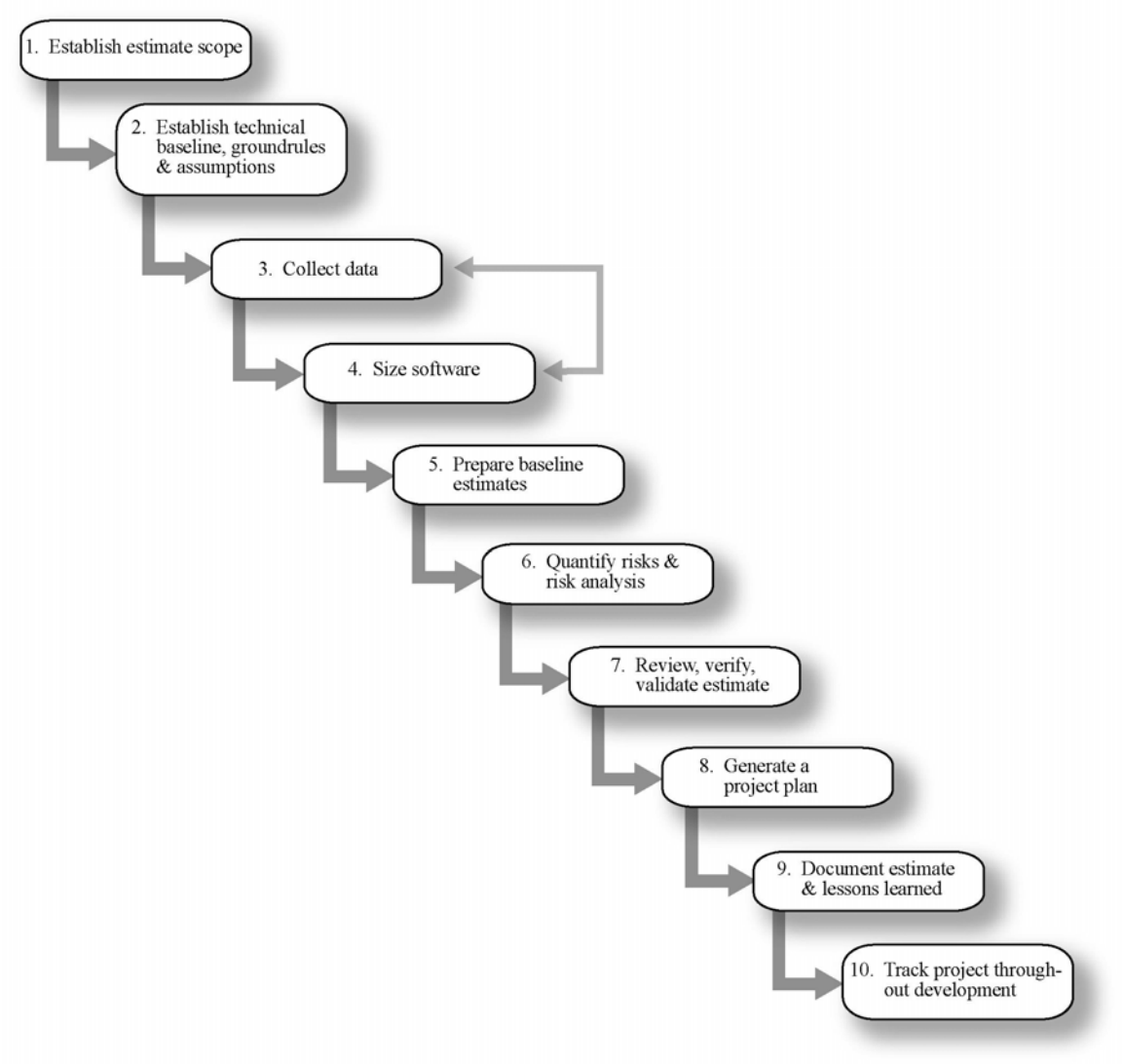


Figure 1 10 Step Estimation Process

Step One: Establish Estimate Scope and Purpose

Define and document estimate expectations. When all participants understand the scope and purpose of the estimate, you'll not only have a baseline against which to gauge the effect of future changes; you'll also head off misunderstandings among the project group and clear up contradictory assumptions about what is expected.

Documenting the application specifications, including technical details, external dependencies and business requirements, will provide valuable input for estimating the

resources required to complete the project. The more detailed the specs, the better. Only when these requirements are known and understood can you establish realistic development costs.

An estimate should be considered a living document; as data changes or new information becomes available, it should be documented and factored into the estimate in order to maintain the project's integrity.

Step Two: Establish Technical Baseline, Groundrules, and Assumptions

To establish a reasonable technical baseline, you must first identify the functionality included in the estimate. If detailed functionality is not known, groundrules and assumptions should clearly state what is and isn't included in the estimate. Issues of COTS, reuse, and other assumptions should be documented as well.

Groundrules and assumptions form the foundation of the estimate and, although in the early stages of the estimate they are preliminary and therefore rife with uncertainty, they must be credible and documented. Review and redefine these assumptions regularly as the estimate moves forward.

Step Three: Collect Data

Any estimate, by definition, encompasses a range of uncertainty, so you should express estimate inputs as least, likely and most rather than characterizing them as single data points. Using ranges for inputs permits the development of a viable initial estimate even before you have defined fully the scope of the system you are estimating.

Certain core information must be obtained in order to ensure a consistent estimate. Not all data will come from one source and it will not all be available at the same time, so a comprehensive data collection form will aid your efforts. As new information is collected, you will already have an organized and thorough system for documenting it.

Step Four: Software Sizing

If you lack the time to complete all the activities described in the ten-step process, prioritize the estimation effort: Spend the bulk of the time available on sizing (sizing databases and tools like SEER-AccuScope can help save time in this process). Using an automated software cost and schedule tool like SEER-SEM can provide the analyst with time-saving tools (SEER-SEM knowledge bases save time in the data collection process).

Size is generally the most significant (but certainly not the only) cost and schedule driver. Overall scope of a software project is defined by identifying not only the amount of new software that must be developed, but also must include the amount of preexisting, COTS, and other software that will be integrated into the new system. In addition to estimating product size, you will need to estimate any rework that will be required to develop the product, which will generally be expressed as source lines of code (SLOC) or function points, although there are other possible units of measure. To help establish the overall uncertainty, the size estimate should be expressed as a least—likely—most range.

Predicting Size

Whenever possible, start the process of size estimation using formal descriptions of the requirements such as the customer's request for proposal or a software requirements specification. You should reestimate the project as soon as more scope information is determined. The most widely used methods of estimating product size are:

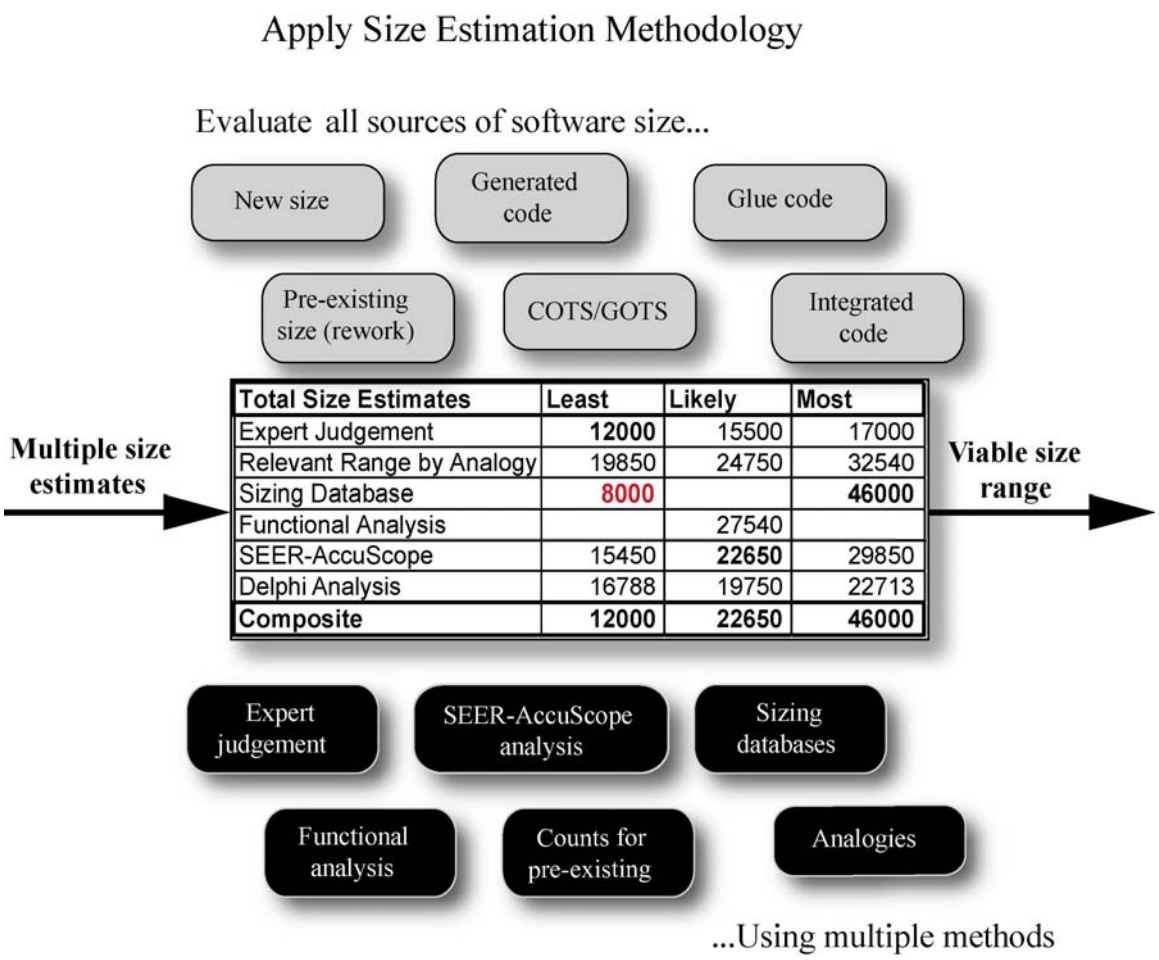
Expert opinion — This is an estimate based on recollection of prior systems and assumptions regarding what will happen with this system, and the experts' past experience.

Analogy — A method by which you compare a proposed component to a known component it is thought to resemble, at the most fundamental level of detail possible. Most matches will be approximate, so for each closest match, make additional size adjustments as necessary. A relative sizing approach such as SEER-AccuScope can provide viable size ranges based on comparisons to known projects.

Formalized methodology — Use of automated tools and/or pre-defined algorithms such as counting the number of subsystems or classes and converting them to function points.

Statistical sizing — Provides a range of potential sizes that is characterized by least, likely, and most.

Use the Galorath sizing methodology to quantify size and size uncertainty. This includes preparing as many size estimates as time permits and putting them all in a table (Figure 2), then choosing the size range from the variety of sources.



Total Size Estimates	Least	Likely	Most
Expert Judgement	**12000**	15500	17000
Relevant Range by Analogy	19850	24750	32540
Sizing Database	8000		**46000**
Functional Analysis		27540	
SEER-AccuScope	15450	**22650**	29850
Delphi Analysis	16788	19750	22713
Composite	**12000**	**22650**	**46000**

Figure 2 Galorath Size Methodology Identifies Size Ranges From Multiple Methods

Step Five: Prepare Baseline Estimate

Budget and schedule are derived from estimates, so if an estimate is not accurate, the resulting schedules and budgets are likely to be inaccurate also. Given the importance of the estimation task, developers who want to improve their software estimation skills should understand and embrace some basic practices. First, trained, experienced, and skilled people should be assigned to size the software and prepare the estimates. Second, it is critically important that they be given the proper technology and tools. And third, the project manager must define and implement a mature, documented, and repeatable estimation process.

To prepare the baseline estimate there are various approaches that can be used, including guessing (which is not recommended), using existing productivity data exclusively, the bottom-up approach, expert judgment, and cost models.

Bottom-Up Estimating: Bottom-up estimating, which is also referred to as “grassroots” or “engineering” estimating, entails decomposing the software to its lowest levels by

function or task and then summing the resulting data into work elements. This approach can be very effective for estimating the costs of smaller systems. It breaks down the required effort into traceable components that can be effectively sized, estimated, and tracked; the component estimates can then be rolled up to provide a traceable estimate that is comprised of individual components that are more easily managed. You thus end up with a detailed basis for your overall estimate.

Software cost models: Different cost models have different information requirements. However, any cost model will require the user to provide at least a few — and sometimes many — project attributes or parameters. This information describes the project, its characteristics, the team's experience and training levels, and various other attributes the model requires to be effective, such as the processes, methods, and tools that will be used.

Parametric cost models provide a means for applying a consistent method for subjecting uncertain situations to rigorous mathematical and statistical analysis. Thus they are more comprehensive than other estimating techniques and help to reduce the amount of bias that goes into estimating software projects. They also provide a means for organizing the information that serves to describe the project, which facilitates the identification and analysis of risk.

A cost model uses various algorithms to project the schedule and cost of a product from specific inputs. Those who attempt to merely estimate size and divide it by a productivity factor are sorely missing the mark. The people, the products, and the process are all key components of a successful software project. Cost models range from simple, single formula models to complex models that involve thousands of calculations.

Delphi and Wideband Delphi: You can further offset the effects of these biases by implementing the Delphi estimation method, in which several expert teams or individuals, each with an equal voice and an understanding up front that there are no correct answers, start with the same description of the task at hand and generate estimates anonymously, repeating the process until consensus is reached.

Activity-Based Estimates: Another way to estimate the various elements of a software project is to begin with the requirements of the project and the size of the application, and then, based on this information, define the required tasks, which will serve to identify the overall effort that will be required.

The major cost drivers on a typical project are focused on the non-coding tasks that must be adequately considered, planned for, and included in any estimate of required effort. Of course, not every project will require all of these tasks, and you should tailor the list to the specific requirements of your project, adding and deleting tasks as necessary and modifying task descriptions if required, and then build a task hierarchy — which usually takes the form of a WBS — that represents how the work will be organized and performed. The resulting work breakdown structure is the backbone of the project plan

and provides a means to identify the tasks to be implemented on a specific project. It is not a to-do list of every possible activity required for the project; it does provide a structure of tasks that, when completed, will result in satisfaction of all project commitments.

Step Six: Quantify Risks and Risk Analysis

It is important to understand what a risk is and that a risk, in itself, does not necessarily pose a threat to a software project if it is recognized and addressed before it becomes a problem. Many events occur during software development. Risk is characterized by a loss of time, or quality, money, control, understanding, and so on. The loss associated with a risk is called the risk impact.

We must have some idea of the probability that the event will occur. The likelihood of the risk, measured from 0 (impossible) to 1 (certainty) is called the risk probability. When the risk probability is 1, then the risk is called a problem, since it is certain to happen.

For each risk, we must determine what we can do to minimize or avoid the impact of the event. Risk control involves a set of actions taken to reduce or eliminate a risk.

Risk management enables you to identify and address potential threats to a project, whether they result from internal issues or conditions or from external factors that you may not be able to control. Problems associated with sizing and estimating software potentially can have dramatic negative effects. The key word here is potentially, which means that if problems can be foreseen and their causes acted upon in time, effects can be mitigated. The risk management process is the means of doing so.

Step Seven: Estimate Validation and Review

At this point in the process, your estimate should already be reasonably good. It is still important to validate your methods and your results, which is simply a systematic confirmation of the integrity of an estimate. By validating the estimate, you can be more confident that your data is sound, your methods are effective, your results are accurate, and your focus is properly directed.

There are many ways to validate an estimate. Both the process used to build the estimate and the estimate itself must be evaluated. Ideally, the validation should be performed by someone who was not involved in generating the estimate itself, who can view it objectively. The analyst validating an estimate should employ different methods, tools and separately collected data than were used in the estimate under review.

When reviewing an estimate you must assess the assumptions made during the estimation process. Make sure that the adopted ground rules are consistently applied throughout the

estimate. Below-the-line costs and the risk associated with extraordinary requirements may have been underestimated or overlooked, while productivity estimates may have been overstated. The slippery slope of requirements creep may have created more uncertainty than was accounted for in the original estimate.

A rigorous validation process will expose faulty assumptions, unreliable data and estimator bias, providing a clearer understanding of the risks inherent in your projections. Having isolated problems at their source, you can take steps to contain the risks associated with them, and you will have a more realistic picture of what your project will actually require to succeed.

Despite the costs of performing one, a formal validation should be scheduled into every estimation project, before the estimate is used to establish budgets or constraints on your project process or product engineering. Failing to do so may result in much greater downstream costs, or even a failed project.

Step Eight: Generate A Project Plan

The process of generating a project plan includes taking the estimate and allocating the cost and schedule to a function and task-oriented work breakdown structure.

To avoid tomorrow's catastrophes, a software manager must confront today's challenges. A good software manager must possess a broad range of technical software development experience and domain knowledge, and must be able to manage people and the unique dynamics of a team environment, recognize project and staff dysfunction, and lead so as to achieve the expected or essential result.

Some managers, mainly due to lack of experience, are not able to evaluate what effects their decisions will have over the long run. They either lack necessary information, or incorrectly believe that if they take the time to develop that information the project will suffer as a result. Other managers make decisions based on what they think higher management wants to hear. This is a significant mistake. A good software manager will understand what a project can realistically achieve, even if it is not what higher management wants. His job is to explain the reality in language his managers can understand. Both types of "problem manager," although they may mean well, either lead a project to an unintended conclusion or, worse, drift down the road to disaster.

Software management problems have been recognized for decades as the leading causes of software project failures. In addition to the types of management choices discussed above, three other issues contribute to project failure: bad management decisions, incorrect focus, and destructive politics. Models such as SEER-SEM handle these issues by guiding you in making appropriate changes in the environment related to people, process, and products.

Step Nine: Document Estimate and Lessons Learned

Each time you complete an estimate and again at the end of the software development, you should document the pertinent information that constitutes the estimate and record the lessons you learned. By doing so, you will have evidence that your process was valid and that you generated the estimate in good faith, and you will have actual results with which to calibrate your estimation models. Be sure to document any missing or incomplete information and the risks, issues, and problems that the process addressed and any complications that arose. Also document all the key decisions made during the conduct of the estimate and their results and the effects of the actions you took. Finally, describe and document the dynamics that occurred during the process, such as the interactions of your estimation team, the interfaces with your clients, and trade-offs you had to make to address issues identified during the process.

You should conduct a lessons-learned session as soon as possible after the completion of a project while the participants' memories are still fresh. Lessons-learned sessions can range from two team members meeting to reach a consensus about the various issues that went into the estimation process to highly structured meetings conducted by external facilitators who employ formal questionnaires. No matter what form it may take, it is always better to hold a lessons-learned meeting than not, even if the meeting is a burden on those involved. Every software project should be used as an opportunity to improve the estimating process.

Step Ten: Track Project throughout Development

In-process information should be collected and the project should be tracked and compared to the original plan. If projects are varying far off their plans refined estimates should also be prepared. Ideally, the following attributes of a software project would be tracked:

1. Cost, in terms of staff effort, phase effort and total effort
2. Defects found or corrected, and the effort associated with them
3. Process characteristics such as development language, process model and technology
4. Project dynamics including changes or growth in requirements or code and schedule
5. Project progress (measuring performance against schedule, budget, etc.)
6. Software structure in terms of size and complexity

Earned value, combined with quality and growth can be used to forecast completion very accurately and flag areas where managers should be spending time controlling.

Summary

Viable software cost, schedule, risk estimation is a difficult process but a necessary part of a successful software development. You can help ensure useful results by adopting a process that is standardized and repeatable. Several of the steps we have discussed, particularly those that do not result directly in the production of the estimate (Steps 1, 6, and 7) are often deferred or, worse still, not performed at all, often for what appear to be good reasons such as a lack of adequate time or resources or a reluctance to face the need to devise a plan if a problem is detected. Sometimes you simply have more work than you can handle and such steps don't seem absolutely necessary. Sometimes management is reluctant to take these steps, not because the resources are not available, but because managers do not want to really know what they may learn as a result of scoping their estimates, quantifying and analyzing risks, or validating their estimates. This can be a costly attitude, because in reality every shortcut results in dramatic increases in project risks.

References

Boehm, Barry. *Software Engineering Economics*. Upper Saddle River: Prentice Hall, 1981.

Boehm, B.W., C. Abts, A.W. Brown, S. Chulani, B. Clark, E. Horowitz, R. Madachy, D. Reifer, and B. Steece. *Software Cost Estimation with COCOMO II*. Upper Saddle River: Prentice Hall, 2000.

Booch, Grady. *The Software Development Team Whitepaper*. Cupertino: Rational Software Corporation, 1999.

DeMarco, Tom. *Controlling Software Projects: Management, Measurement, and Estimation*. Englewood Cliffs: Yourdon Press, 1998.

DeMarco, Tom. *Why Does Software Cost So Much?* New York: Dorsett House, 1995.

DeMarco, Tom and Tim Lister. *Peopleware; Productive Projects and Teams*, 2nd ed. New York: Dorsett House, 1999.

Evans Michael, Alex Abela, and Tom Beltz. "Seven Characteristics of Dysfunctional Software Projects." *CrossTalk: The Journal of Defense Software Engineering*, April 2002.

Ferens, Daniel, L. Fischman, T. Fitzpatrick, D. Galorath, and D. Tarbet. "Automated Software Project Size Estimation via Use Case Points." Report to the U.S. Government, January 2002.

Florac, William A., Robert Park, and Anita D. Carleton. *Practical Software*

Measurement: Measuring for Process Management and Improvement. Pittsburgh: Carnegie Mellon Software Engineering Institute, 1997.

Galorath, Daniel. “Software Productivity Laws.” Arthur Anderson Symposium, 1997.

Galorath, Daniel, Evans, Michael, “Software Sizing, Estimation, and Risk Management: When Performance Is Measured Performance Improves”, Auerbach Publishing, 2006)

Galorath, Daniel, Lee Fischman, and Dan Ferens. “Critical Mass: Advancing the Software Sizing State of the Art, Progress and Lessons Learned.”

Galorath Incorporated. *SEER-SEM Internal Mathematical Specification*. El Segundo, 2004.

Humphrey, Watts. “Three Dimensions of Process Improvement. Part I: Process Improvement.” *CrossTalk: The Journal of Defense Software Engineering*. February 1998.

Humphrey, W.S., T.R. Snyder, and R.R. Willis. “Software Process Improvement at Hughes Aircraft.” *IEEE Software*, July 1991.

International Society of Parametric Analysis. *Parametric Estimating Handbook*, 2nd ed. Sponsored by the U.S. Department of Defense. Chandler, 2002. <http://www.ispa-cost.org/PEIWeb/newbook.htm>

Jones, Capers. *Estimating Software Costs*. New York: McGraw-Hill, 1998.

Jones, Capers T. *Assessment and Control of Software Risks*. Englewood Cliffs: Prentice Hall, February 1994.

Park, Robert E. *A Manager’s Checklist for Validating Software Cost and Schedule Estimates*. Pittsburgh: Carnegie Mellon Software Engineering Institute, January 1995.

Putnam, Lawrence H., and Ware Meyers. *Industrial Strength Software, Effective Management Using Measurement*. Washington, D.C.: IEEE Computer Press, 1997.

Reifer, Donald J. *Practical Software Reuse*. 1997.

Stutzke, Richard D. *Estimating Software Intensive Systems: Projects, Products, and Processes* (SEI Series in Software Engineering), 2005.

Bio:

Daniel D. Galorath is CEO and Founder of Galorath Incorporated, and is a recognized expert in software estimation and sizing. During his over three decades in the computer industry, Mr. Galorath has been solving a variety of management, costing, systems, and software problems and has performed all aspects of software development and management. His company, Galorath Incorporated, has developed tools, methods, and training for software cost, schedule, risk analysis, and management decision support, including the industry standard SEER-SEM™ software evaluation model. Dan Galorath completed his undergraduate and MBA, both from California State Universities.