Size Does Matter:
Continuous Size Estimating and Tracking

Mike Ross
Quantitative Software Management, Inc.
5013 W. Vogel Ave.
Glendale, AZ 85302
(623) 435-9863 (phone) (623) 915-3351 (fax)
mike_ross@qsm.com
http://www.qsm.com

Abstract. Estimating the size of a software system is a critical development process activity. Not only does size impact the *technical* solution; it also impacts the *project management* solution. It is therefore insufficient to estimate size only once, at the beginning of the project when the least is known about the system being developed. This paper describes a quantitative process for managing the size of software development projects through continuous estimation. The process includes a probabilistic approach to estimation coupled with tracking and assessment of trends to determine whether or not stability exists. This process has direct applicability to the SEI CMM Level 2 KPAs for Software Project Planning and Software Project Tracking and Oversight. It also serves as a Level 4 Quantitative Process Management tool for measuring the effectiveness of an organization's size estimating and requirements management processes. [3], [4], [5]

Introduction

Purpose

The purpose of this paper is to justify and describe a process for quantitatively managing the size of software development projects through continuous estimation.

Scope

The process described in this paper applies to all software development projects. Within the scope of the SEI CMM, this process applies to the Level 2 KPAs for Software Project Planning and Software Project Tracking and Oversight. It also serves as a Level 4 Quantitative Process Management tool for measuring the effectiveness of an organization's size estimating and requirements management processes. [3], [4], [5]

Background

> *Estimating size is the heart of the software-project estimating process.*
> Lawrence H. Putnam [6]

A casual glance at any software project estimating model illustrates the fundamental truth of the above statement. It follows, and historical data bears this out, that many software projects fail to meet cost, schedule, and reliability expectations because the actual size ends up being much larger than expected. Why, then, do so many organizations estimate size only once, at the beginning of the project when the least is known about the system being developed? These same organizations tend to revisit a size estimate only if, and more importantly when, the project is hopelessly out of control.

Intuition, Research, and Supporting Data

What is "Size"?

> *There are two, often-confused, notions of software size: one that relates to work and one that relates to functionality.*

The modern computing environment poses many challenges. Foremost among them is addressing totally new technologies. Trying to figure out how big computer programs are has challenged software engineering since its inception and is further complicated by the aforementioned dynamic nature of technology. [1]

Analysts have traditionally sized systems written in statement-oriented procedural languages expressed largely as text (large stacks of cards or reams of tractor-feed paper). Current technologies now take the form of more abstract representations such as diagrams, objects, spreadsheet cells, database queries, and Graphical User Interface (GUI) widgets. [1]

The secret to making progress in sizing these new environments is to identify the $unit\ of\ human\ thought$ in the abstraction being used. Next, the organization must go through a calibration process, starting with projects for which the actual size can be determined in terms of that $unit\ of\ human\ thought$. The goal of calibration is to establish productivity as a function of actual size, actual time, and actual effort for completed projects. This newly-established productivity relationship can then be used to fine-tune the sizing process and to forecast time and effort on a new project. Once the project is complete, another iteration of the calibration process can be done. This cycle, repeated numerous times, yields sizing and forecasting methods that exhibit a high degree of accuracy with minimal variation. [1]

Sizing is one of the hardest things a development organization does, and the earlier in the life cycle it is done, the harder it is to do. [1]

Historically, statements (Source Lines of Code or SLOC) have been used for sizing systems, sometimes with poor results due to the difficulty of making the mental leap across the “abstraction chasm” from operational capability to programming language constructs. SLOC, however, is an excellent measure of the “work” done by the development process and is most effectively used in process productivity metrics. Advantages: 1) it can be unambiguously defined for a given language; 2) measuring the size of an existing product is automatable; 3) most of the world’s historical data contains SLOC as the sizing measure. Disadvantages: 1) the notion of SLOC becomes ambiguous when dealing with non-textual abstractions; 2) the measure has little meaning to the customer / end user. [1]

Function Points (FP) offer a way of narrowing the “abstraction chasm” by providing a level of abstraction between operational capability and programming language constructs. FP is an excellent measure of the ”functionality” or “value” produced by the development process and is most effectively used in “bang for the buck” type metrics. Advantages: 1) the customer / end user can likely relate to the entities being counted; 2) there are networks of people (e.g., IFPUG) dedicated to standardizing and improving the counting process. Disadvantages: 1) FP are limited to application domains for which their countable entities make sense (typically mainframe business applications); 2) the process of counting the number of FP in a finished product is not automatable, in fact many big FP shops do quick-and-dirty estimates, using shortcuts such as “backfiring” (back-calculating FP as a function of language and size in SLOC). [1]

A major qualifier on the use of FP in a productivity relationship is the fact that FP do not directly relate to development process “work” and must be scaled as a function of the programming language used. This scaling introduces an additional source of complexity and variability in the relationship. [1]

If all this isn’t enough to complicate the selection of sizing measures, consider that many new development methodologies employ abstractions that are neither textual nor do their components fit within the set of FP counting entities. [1]

Size Drives Cost, Schedule, and Reliability

> *If you underestimate the size of your next project, common sense says that it doesn’t matter which methodology you use, what tools you buy, or even who you assign to the job.*
> Ed Yourdon

Historical data shows that increasing the size of a system will increase its cost and schedule, and reduce its reliability at delivery (see Figure 1, Figure 2, and Figure 3). [6]

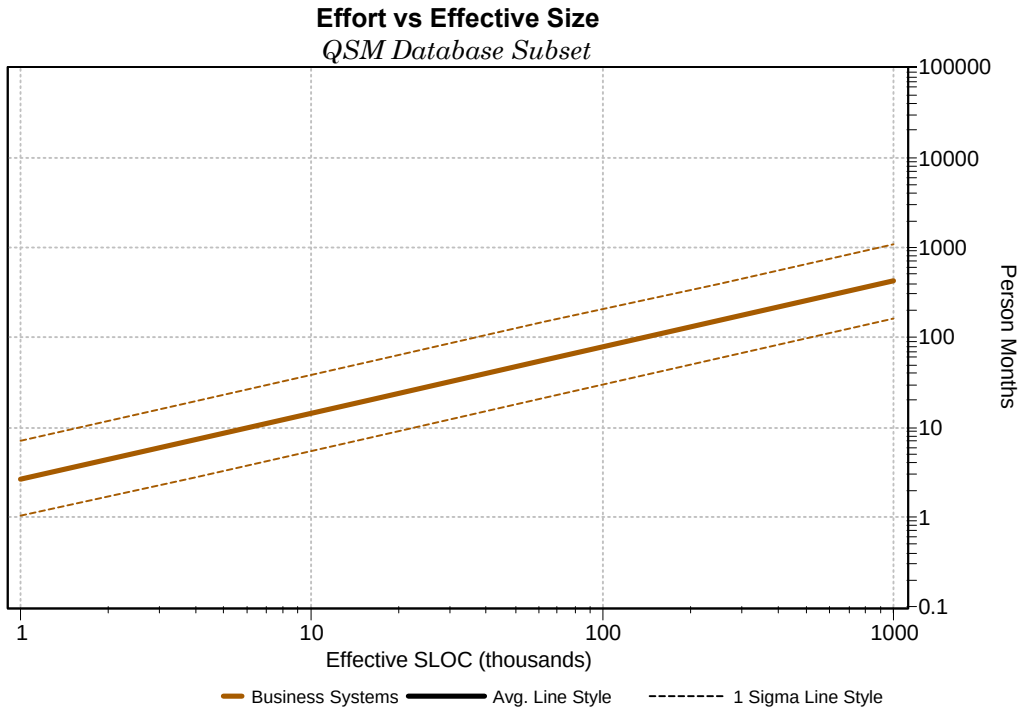

Figure 1: Effort (Cost) Increases with Size

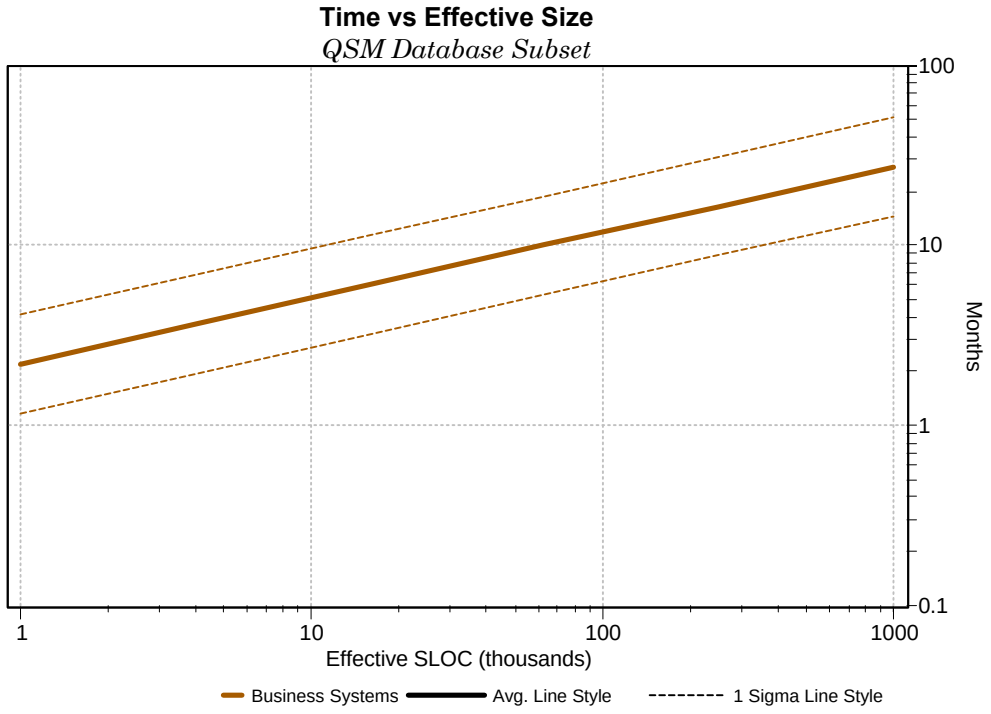

Figure 2: Schedule Increases with Size

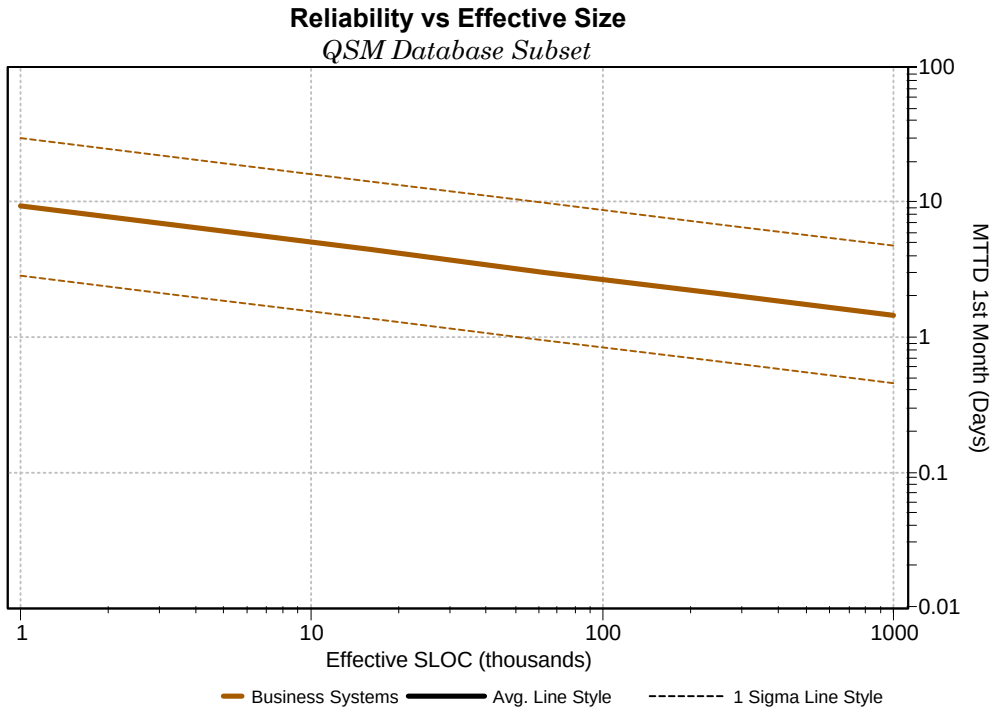

Figure 3: Reliability Decreases with Size

Size Estimation is Probabilistic

> *[Unfortunately, most estimates are] the most optimistic prediction that has a non-zero probability of coming true.*
> Tom DeMarco [2]

Size estimation is a probabilistic problem. This can be verified by simply analyzing the language that people use when they are asked to estimate something. Words and phrases such as *about*, *probably*, *somewhere around*, and *in the neighborhood of*, may seem, to some as weaseling or hedging; however, what they really indicate is the presence of uncertainty. Far too often, the tendency is to produce an estimate that represents the best case scenario.

Knowledge Increases with Time

> *In the beginning a software project is little more than a gleam in one person's eye. Yet his organization may need rough estimates of the cost and schedule to fit into advanced budgets for the next several years. As work on the concept proceeds, more becomes known about it and more precise estimates become possible.*
> Lawrence H. Putnam & Ware Myers [6]

Knowledge about any system being developed increases with time. Customers get more focused about the system's requirements. Developers get smarter about the technologies involved. Common sense suggests that as a project approaches completion:

- The rate of change of a system's estimated final size should approach zero.
- The uncertainty about a system's estimated final size should approach zero.

Change is Inevitable

> *"The only thing constant in the universe is change."*
> Unknown

A rare project experiences no requirements change or scope growth. Figure 4 illustrates that over 60% of all projects experience at least a 10% growth in requirements. A qualitative examination of the underlying data shows requirements growth to be perceived as a dominant negative factor. Intuition and experience suggest that the impact of a requirement change is inversely related to the time remaining in the schedule. In other words, requirements changes that occur late in the project have more impact than do those occurring early in the project.

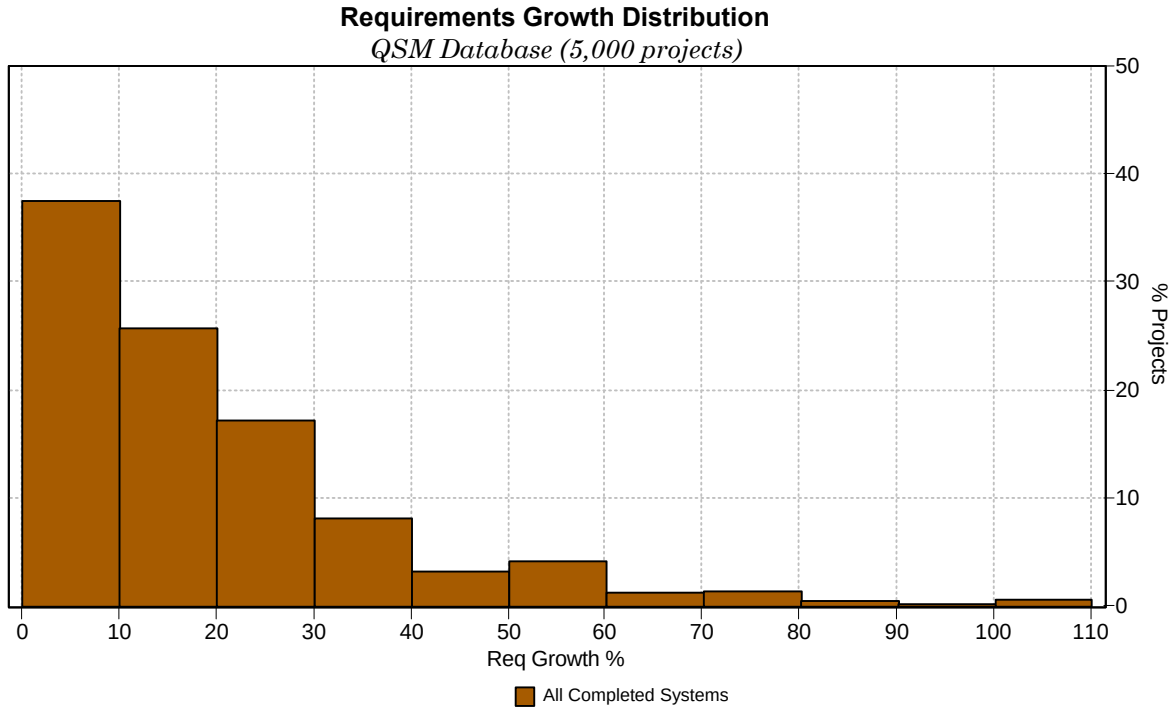

Figure 4: Requirements Growth Distribution

Developers are Optimists

> *Most people are congenitally optimistic (you have to be to get out of bed in the morning).*
> Ware Myers [7]
>
> *When you care a lot about the result, the quality of your estimate approaches zero.*
> Tom DeMarco [2]

Members of the development team are typically the ones called upon to provide estimates for the size of their systems. This would seem to make sense; after all, they know the most about what's being developed. The problem is, these same people are biased toward optimism by having a stake in the project's outcome. Generally, when someone is asked to provide an estimate of an outcome for which they are directly responsible, the tendency is to provide a best-case response.

Process Description

> *"If you don't measure, then you're left with only one reason to believe that you are still in control: hysterical optimism."*
> Tom DeMarco [2]

Estimate Periodically and Record the Data

A fundamental aspect to effectively managing the size of a system is to periodically re-estimate its size and to record the results. Figure 5 shows an example of an artifact for logging periodic size estimates.

Edit Estimate Data for ESLOC (CUM)

Start Date	to	End Date
Jan-96		Dec-97

Month	Low	Most Likely	High	Expected	Convergence
Jan-96	45000	60000	75000	60000	0.00
Feb-96	45000	60000	75000	60000	0.00
Mar-96	45000	60000	75000	60000	0.00
Apr-96	45000	60000	75000	60000	0.00
May-96	45000	60000	75000	60000	0.00
Jun-96	53000	62000	70000	61833	43.33
Jul-96	55000	62500	70500	62583	48.33
Aug-96	56000	63000	71000	63167	50.00
Sep-96	57500	63500	72500	64000	50.00
Oct-96	58000	64000	74000	64667	46.67
Nov-96	59000	64500	76000	65500	43.33
Dec-96	60000	65000	78000	66333	40.00
Jan-97	62000	66500	81500	68250	35.00
Feb-97	65500	68000	84000	70250	38.33
Mar-97	67500	70000	86500	72333	36.67
Apr-97	70500	73000	88500	75167	40.00
May-97	72500	75000	90000	77083	41.67
Jun-97	75500	78000	91500	79833	46.67
Jul-97	78500	81000	92500	82500	53.33
Aug-97	82500	85000	93500	86000	63.33
Sep-97	86500	89000	94500	89500	73.33
Oct-97	89500	92000	95000	92083	81.67
Nov-97	91500	94000	95500	93833	86.67
Dec-97	92500	95000	96000	94750	88.33

Figure 5: Sample Data Entry Form

Express Estimates as Ranges

An estimate, to be of any real value, must include two components: *magnitude* and *uncertainty*:

> ***Magnitude*** quantifies the most likely (best guess) outcome. Probabilistically speaking, this means that 50% of the time the actual outcome will be lower than the most likely outcome and 50% of the time the actual outcome will be higher than the most likely outcome.
>
> ***Uncertainty*** quantifies the distribution of all possible outcomes. This can be expressed as a low outcome and a high outcome between which 99% of all possible outcomes will fall.

An estimate expressed in this form is hereinafter referred to as a *3-point estimate*.

Adjust for Bias

Since most projects experience requirements growth, it follows that the difference between the $most\ likely$ and $high$ values in a 3-point estimate is usually much larger than is the difference between the $most\ likely$ and low values. To dampen the effect of this high-side bias, the notion of an $expected$ value is introduced as follows:

$$\text{Expected} = \frac{\text{Low} + 4 \times \text{Most Likely} + \text{High}}{6} \text{ [6]}$$

Viewing a 3-point estimate as defining some associated normal distribution, one standard deviation can then be approximated by the following:

$$\text{Standard Deviation } (s) = \frac{\text{High} - \text{Low}}{6} \text{ [6]}$$

Ideally, the expected size estimate value would remain constant throughout the project; i.e., the baseline value equals final actual value. This rarely happens; however, since it is the goal, this constancy forms the basis for the plan and variance threshold values in the tracking methodology.

Look for Convergence

Since the uncertainty about a system's estimated final size should approach zero, we need to establish a way to quantify and track this decreasing uncertainty. Here we introduce the notion of convergence. As knowledge and experience increase, uncertainty decreases; therefore, the high size estimate and low size estimate curves should converge. For a given elapsed calendar time t, convergence C_t is a function of the corresponding high H_t and low L_t estimates as follows:

$$C_t = 1 - \frac{H_t - L_t}{H_{Baseline} - L_{Baseline}}$$

Convergence in this context is represented as a percentage:

- A value of 0% (0) convergence represents the level of uncertainty contained in the baseline 3-point estimate.
- A value of 100% (1) convergence represents the total absence of uncertainty (i.e., the high size estimate and the low size estimate are equal).

Note that it is possible to have negative convergence (divergence) which happens when the current uncertainty is greater than the baseline uncertainty.

Cumulative effort (cost), cumulative code production, and cumulative defect discovery all behave according to the cumulative form of a Rayleigh distribution [6] (sometimes referred to in project management circles as an S curve). Uncertainty is a function of the problem-solving process, people solve problems, and people are applied to projects in a Rayleigh-like pattern. [6] A reasonable inference then is that, ideally, size estimate convergence

should go from 0% at the beginning of the project and approach 100% by the end of the project according to the cumulative form of the Rayleigh function. This ideal behavior forms the basis for the plan and variance threshold values in the tracking methodology.

Track Over Time

Tracking of periodic 3-point size estimates provides valuable comparison and trend information. The idea is to track quantities that can easily verify expected project behavior and have the best chance of providing early warning when the project is in trouble. As a reasonable minimum, track *expected size estimate* and *size estimate convergence* as functions of elapsed calendar time. Additionally, add variance thresholds to the graphs to support the process control rules described in the next section (as exemplified by Figure 6, Figure 7, Figure 8, and Figure 9).

Control the Process

Within the context of quantitative project management, control means comparing the desired (planned) outcome to the measured (actual) in-process outcome and basing any corrective action on the difference. Plan the project and take action that causes the earliest possible convergence on what will be the final actual size value.

The following is an example implementation of a size-estimate-based control process that uses observations from the time-based tracking curves and a *traffic light* metaphor with associated rules.

Green Status—No corrective action recommended.

- The default state; i.e., the criteria for neither *Yellow Status* nor *Red Status* have been met.

Yellow Status—Determine the cause, take corrective action.

- Current *Expected Size Estimate* is outside ±1σ of the baseline 3-point estimate ***or***
- Current *Size Estimate Convergence* is outside ±1σ of the baseline plan Rayleigh convergence curve.

Red Status—Determine the cause, take corrective action, replan the project based on current 3-point estimate.

- Current *Expected Size Estimate* is outside ±2σ of the baseline 3-point estimate ***or***
- Current *Size Estimate Convergence* is outside ±2σ of the baseline plan Rayleigh convergence curve.

Examples of Application

Project with Stable Sizing

Figure 6 and Figure 7 illustrate the tracking of estimated size data for a project with stable sizing. After some early requirements misunderstandings, the project was re-planned during the fifth month. After the replan, the project experienced minor size growth that was well within the green threshold. The bulk of this growth occurred relatively early in the project; late changes were rare. Also, after the replan, size estimate convergence occurred close to plan and well within the green threshold. Even with the size growth, the project still managed to come in on time and within budget.

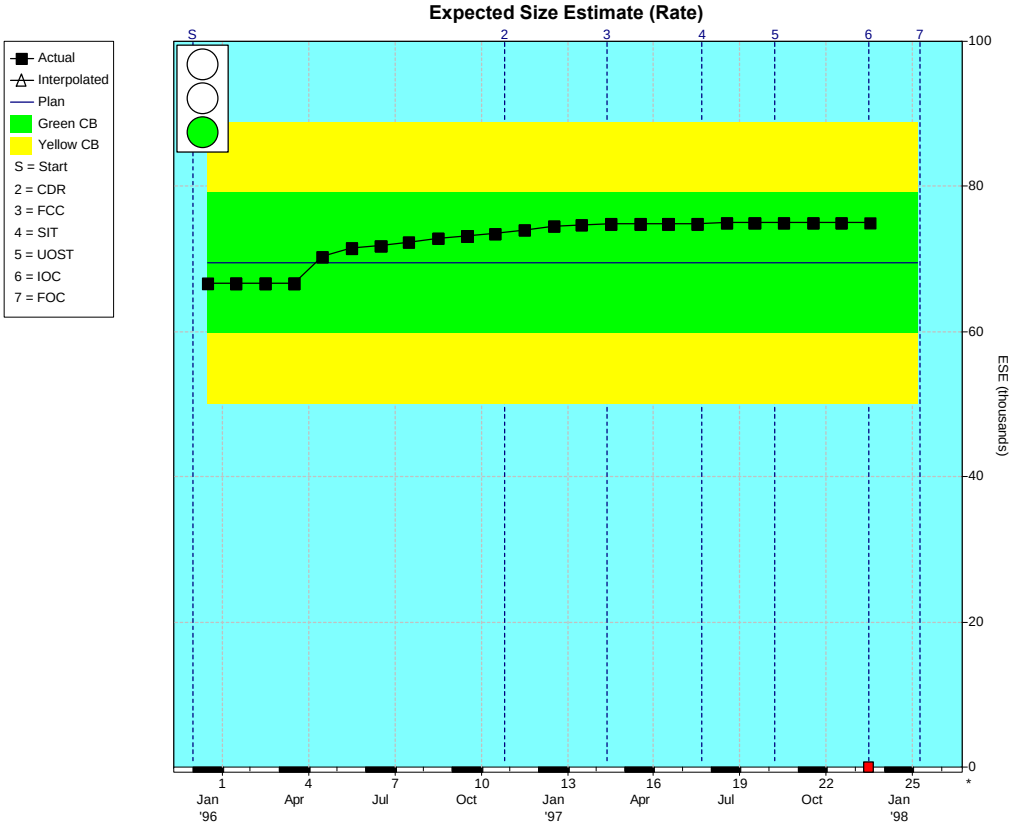

Figure 6: Expected Size Estimate Trend for Project with Stable Sizing

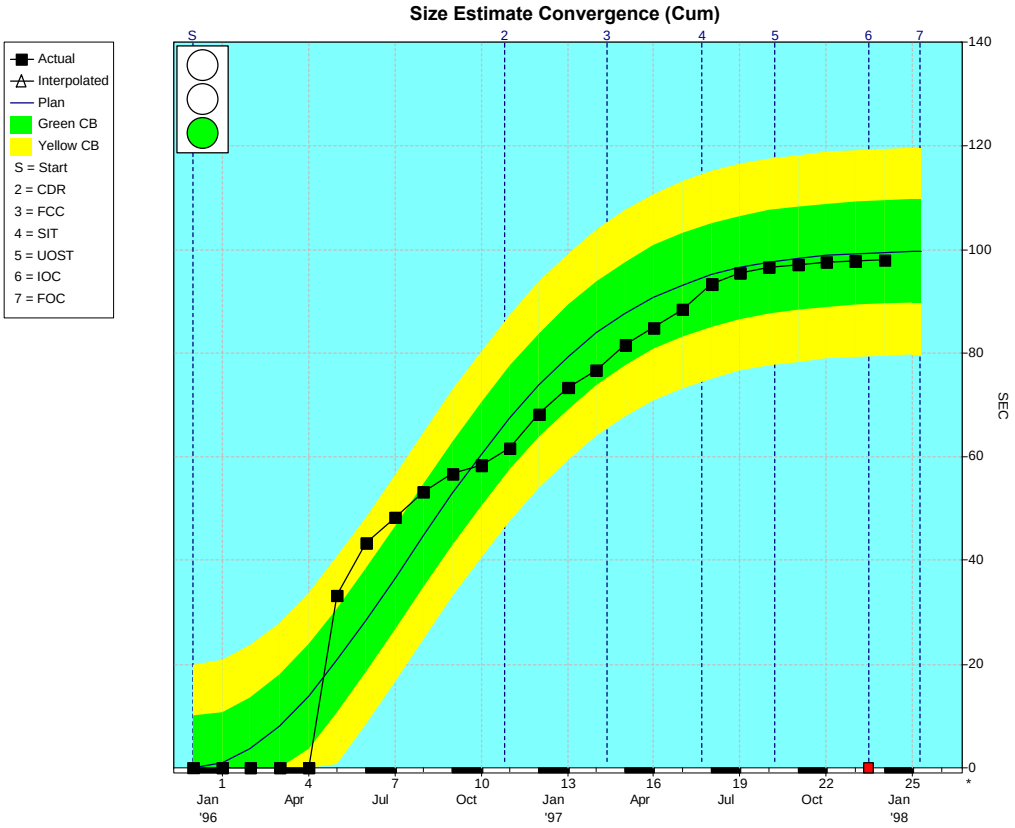

Figure 7: High-Low Size Estimate Convergence for Project with Stable Sizing

Project with Unstable Sizing

Figure 8 and Figure 9 illustrate the tracking of estimated size data for a project with unstable sizing. After some early requirements misunderstandings, the project was re-planned during the fifth month. Unfortunately, after the tenth month, some serious uncertainty began to creep in. The project was hit with several new requirements plus it lost a substantial amount of expected reuse benefit. Uncertainty increased as the bad news continued to roll in. Surprisingly, no replan was done and no get-well strategy was implemented. The expected size continued to grow, unchecked, until the customer finally initiated a code freeze and took delivery of the system as it was. The supplier subsequently lost in a competitive bid for the next phase of the program.

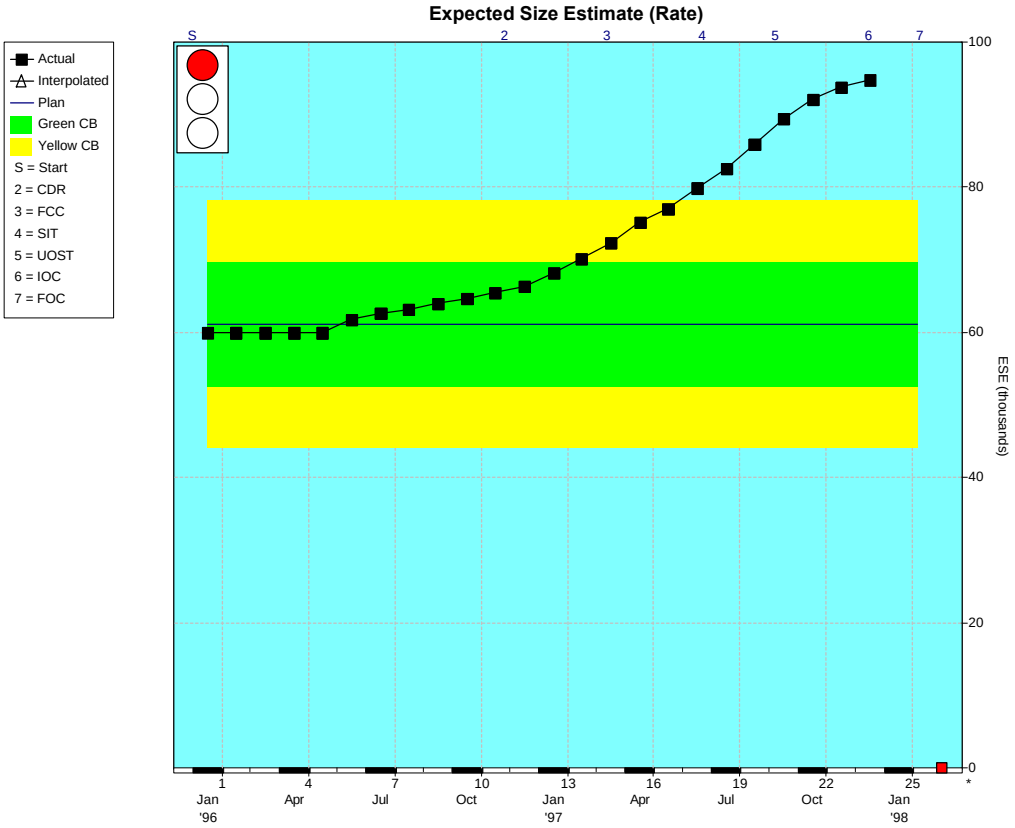

Figure 8: Expected Size Estimate Trend for Project with Unstable Sizing

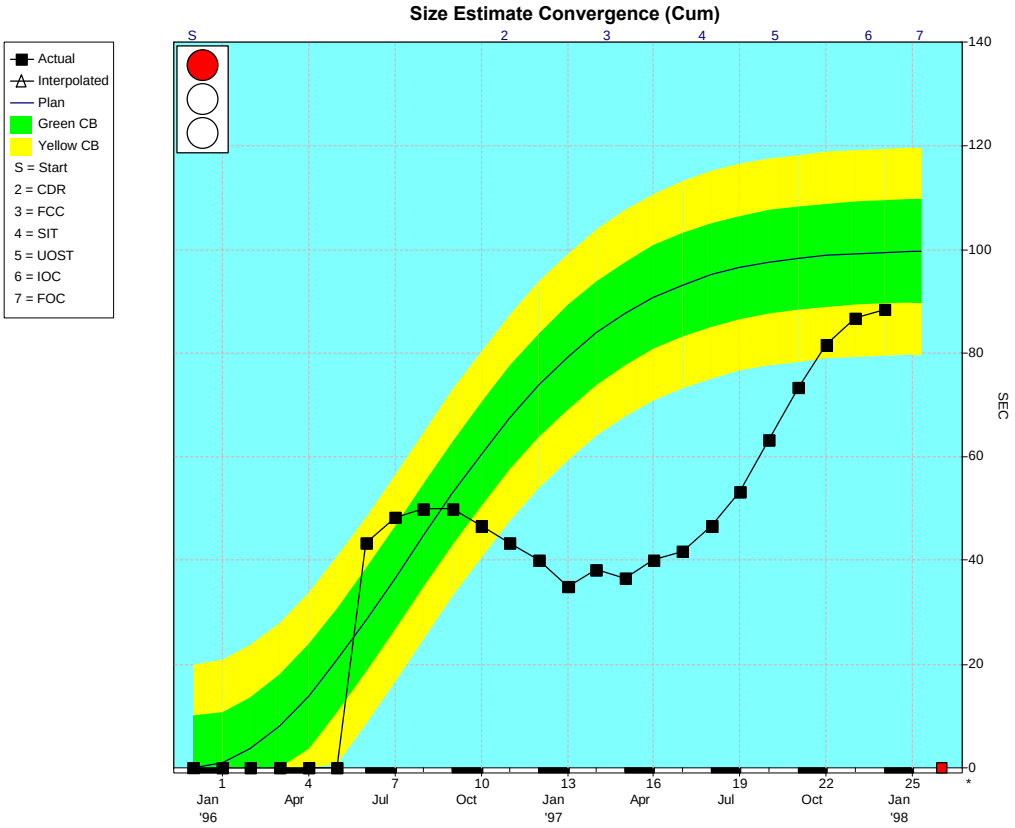

Figure 9: High-Low Size Estimate Convergence for Project with Unstable Sizing

Conclusion

Software size is a quantity that has a profound effect on development cost, schedule, and reliability. Quantitative management of software size is therefore necessary to ensure adequate quantitative management of cost, schedule, and reliability. Since software size is an estimated quantity (i.e., the final value cannot be measured) until the end of development, one must apply quantitative management techniques to the estimates themselves. This can be accomplished with continuous 3-point size estimation and time-based tracking of values derived from these 3-point estimates.

Project management success means achieving expectations. Unreasonable expectations lead to failure. Guard against expectations being too high and too low. Finally, you can't control what you can't measure. [2]

References

[1] Butler, J. & Ross, M., T., "Making the First Cut: Sizing New Technology." *QSM Perspectives*, Fall 1997, Volume 20 Number 2, pp. 1-2, 4.

[2] Demarco, T., *Controlling Software Projects: Management, Measurement, and Estimation.* New York, NY: Yourdon Press, 1982.

[3] Humphrey, W., *Managing the Software Process.* Reading, MA: Addison-Wesley Publishing Co., 1989.

[4] Paulk, M.C., Curtis, B., Chrissis, M.B., Weber, C.V., *Capability Maturity Model for Software, Version 1.1.* Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1993.

[5] Paulk, M.C., Weber, C.V., Garcia, S.M., Chrissis, M.B., Bush, M., *Key Practices of the Capability Maturity Model, Version 1.1.* Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1993.

[6] Putnam, L. & Myers, W., *Measures for Excellence: Reliable Software On Time, Within Budget.* Englewood Cliffs, NJ: Yourdon Press, 1992.

[7] Putnam, L. & Myers, W., *Industrial Strength Software: Effective Management Using Measurement.* Los Alamitos, CA: IEEE Computer Society Press, 1997.

Biography

Michael A. Ross is currently Managing Director of the Western Region Office of Quantitative Software Management, Inc. where, for the last four years, he has served as one of QSM's primary consultants and analysts working with Fortune 500 companies and government agencies in the areas of measurement, sizing, forecasting, and control.

Mr. Ross, during 17 years with Honeywell Air Transport Systems (formerly Sperry Flight Systems), developed or managed the development of embedded software for avionics systems installed in the Lockheed L1011-500, Boeing 757/767, Airbus A320, Douglas MD-11, British Aerospace BAe-146, and Boeing 777 airplanes. He also co-founded the division's process improvement team (later to become its SEPG), served as a corporate SEI CMM assessor, and served as the division's focal for software project management process improvement.

Mr. Ross did his undergraduate work at the United States Air Force Academy and Arizona State University, receiving a Bachelor of Science in Computer Engineering. He is a member of the IEEE, the International Function Points Users Group, the International Society of Parametric Analysts, the Arizona Software Association, and the Phoenix area Software Process Improvement Network.