Quality Metrics and Sizing

No part of technology efforts tends to be more elusive than determining their true scope. There are five primary sizing variables for technology projects. These sizing variables are organized into two groups—base variables and derived variables.

There are two base variables: **mass** (expressed in units of lines of executable source code, function points, source operators and operands, requirements, packages, etc.), and **risk** (expressed in dimensionless units and treated as a proportionality function for scaling each derived variable). These two variables are referred to as base variables because they represent inherent attributes of the solution and its relevant delivery environment.

The mass variable is an attempt to establish the intrinsic magnitude of the solution itself. It is a measure of the information and functionality content of the solution. Solutions that have greater functionality are, in effect, more massive, than solutions with less inherent content. That is, they intuitively require more power to "move" them from one state to another, or to change implementation direction or tempo.

The risk variable is a measure of the complexity of the solution and the reliability of the delivery environment and its relevance and fit with the solution being contemplated. In other words, risk is an indication of how well suited the overall environment (e.g., people, skills, process, tools, organization, etc.) is to the solution at hand. The more suited, the lower the risk; the less suited, the greater the risk.

Mass and risk are critical variables for a project manager to deeply understand because they drive the computation of the values for the three derived variables: **effort** (expressed in units of person hours, person years), time (units of calendar days, weeks), and **cost** (expressed in $).

Moreover, the relationship among these five variables for any given project is decidedly nonlinear, thus dramatically complicating their use in project sizing, and in understanding how changes in one variable affect the values of the other variables. For example, several researchers (Walston and Felix of IBM, and Boehm of TRW) have found that the following equation fits many of the relevant software development models that they studied:

$$effort = k \cdot mass^{f}$$

Where effort is expressed in person months, mass is expressed in lines of code, and k and *f* are empirical factors (similar to our risk variable) ranging from k=5.2 and *f*=.91 for the IBM model to k=3 and *f*=1.12 for one version of the TRW model. Ignoring for a moment the very real counting issues related to the line-of-code choice of the mass unit (e.g., it is source code language dependent, etc.), one can easily see that sizing the lines of code for a solution that is still in the early planning and design stages is a daunting task with enormous variability potential.

Fred Brooks in his excellent collection of software engineering essays illustrates another aspect of this non-linearity by pointing out that "men" (i.e., effort) and months (i.e., time) are not interchangeable. Many software development activities share this property: Adding effort does not necessarily result in a corresponding decrease in schedule. His aphorism that *"It takes a woman nine months to have a baby, no matter how many men are assigned to the job."* is a particularly poetic way of describing this phenomenon.

Brooks goes on to summarize this nonlinear relationship between effort and time with one of his most famous quotes: *"Adding more people to a late software project makes it later."* In this case, the increase in time (rather than the decrease or preservation of the current schedule that was intended) is due to the substantial increase in interaction and learning curve effects as each new team member enters the project, especially late in the effort where team size is largest. These effects tend to be roughly proportional to n^2, where n is team size (actually, they are proportional to $n^{(2)}/2 = (n \cdot (n-1))/2$). In other words, if additional people will be needed, then the earlier they are added, the better. Many project managers have ignored this observation at their considerable peril.

There is an entire industry segment devoted to tools and techniques for assisting project managers in computing these five variables for a given project. While these tools can be very useful, the most reliable sizing methods are those based on direct, quantitative experience in the exact environment, and with the same talent, as the proposed effort.

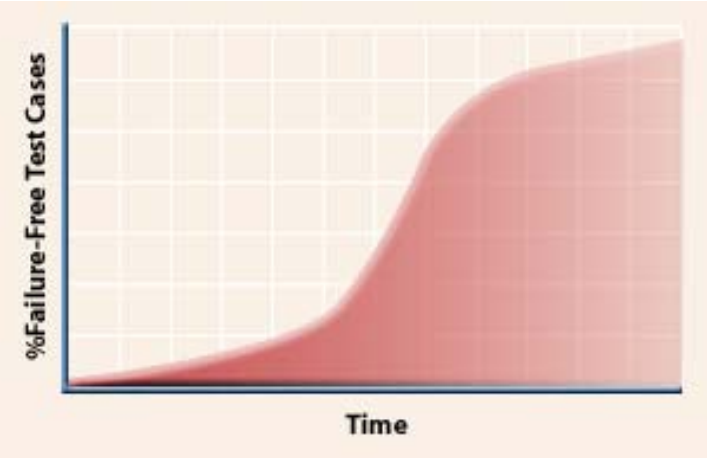
Precise matches of this type are typically impractical, so proxies must be used. The most reliable proxies are quantitative metrics captured by the organization managing the project. These metrics, in effect, characterize the *performance capacity* of a particular software engineering environment:

© 2004 iTest Quality Partners, Inc. All rights reserved.

- **Productivity** (mass/effort, mass/cost) is a measure of the efficiency with which usable functionality can be produced; unit costs are simply the inverse of the corresponding productivity metric
- **Cycle Time** (time/mass) is a measure of the environment's latency or the elapsed time between deliveries; the delivery velocity is the inverse of cycle time

In addition to these primary metrics, there is a set of secondary metrics that is exceedingly useful in better understanding the operational characteristics of the environment as well as the completion status of the project:

- **Cost-To-Complete** (*Rem To Do* for all tasks in WBS) is a measure of work remaining; the change in this value per unit of applied effort is a useful indicator of the current efficiency of the project
- **Earned Value** (validated mass/total mass) is a measure of completeness, where validated mass refers to those components of the solution that have met a predefined quality threshold; due to its emphasis on delivered value from a customer's point of view rather than simply effort expended, this can be a more insightful metric into actual progress
- **Solution Flux** ((current mass - original mass)/original mass) is a measure of the volatility of the solution and provides useful insights into the stability of both the problem and the solution
- **Variance Volatility** (rate of change of current period effort variance) is a measure of the reliability of the project plan and its estimates—the lower the volatility, the greater the reliability
- **Defect Density** (defects/mass) is a measure of the process quality of the packages that make up the technology solution; defects tend to cluster, so that the likelihood of finding the second defect in any given section of source code, for example, increases
- **Failure Intensity** (failures/time) is a measure of the efficacy of the validation process; since the goal of testing is to find problems, not prove something works, this metric can be a useful window into the effectiveness of the test case planning and design process
- **Cumulative Validation Rate** (% failure-free test cases) is a measure of the completeness of the validation process; this metric exhibits the so called S-curve shape (Figure 1), and can be a very helpful tool to understand how much testing remains
- **Defect Removal Efficiency** (defects before ship/total defects) is a measure of the effectiveness of the entire software delivery approach; a minimum of two to three years of production operation is typically required before this metric reveals the true performance of the technology delivery process
- **Design Coverage** (requirement specification **x** module) is a measure of the degree to which the requirements are supported by the functionality of the proposed solution; this metric (and its underlying correlation table) can be very helpful during the early stages of package elaboration in identifying design gaps, over-designed solutions, and related architecture problems before substantial effort has been spent on constructing the solution
- **Validation Coverage** (test case **x** requirement specification) is a measure of the degree to which the requirements are being validated by the test plans; this metric (and its underlying correlation table) is essential in effective test plan and test case design since it ensures that the minimum number of test cases have been devised that will validate the maximum number of requirements



An effective project manager recognizes the importance of quantitative data in understanding the dynamics of the project and in using this fact base to learn, stabilize, and continuously improve the performance capacity of the project environment.

For more information, please contact us at (630) 365-1606, or visit www.itestqp.com.

Figure 1. When is testing done?

© 2004 iTest Quality Partners, Inc. All rights reserved.