Control the Software Beast With Metrics-Based Management

Lawrence H. Putnam
Quantitative Software Management, Inc.

Ware Myers
Consultant

Living on a finite planet with a day length regulated by a sun implies that we have to complete the following: a quantity of work (measured by size), at some level of quality, within limits of time, and effort, at some degree of productivity. These are the five core metrics that enable managers to estimate and bid software projects, and control progress during the project. Higher executives and clients need to understand this pattern because, otherwise, they are the prime source of unrealistic expectations and the resulting failures.

Software development is a *beast* for the many organizations whose project failures keep baffling senior management and clients. But it is a beast that can be tamed. In fact, hundreds of organizations have already done so. Earlier disciplines show the way.

Executives control whole businesses with financial information collected and analyzed by accountants. Manufacturing executives cost factory production with data supplied by cost accountants. It follows then that software projects can be managed with appropriate information – progress indicators applied through metrics-based management.

Beyond the project, however, software development involves a broad range of stakeholders – client management, users, outsourcing organizations, and high-level management. As a result, not only must we establish measures and their collection and accessibility, but we also must communicate the nature of metrics-based management to these stakeholders.

Core Measures Are the Foundation

The measures underlying effective software management are effort, time, size, and a measure of quality or reliability such as the defect rate (or its reciprocal, mean time to defect). These key measures need to be clearly defined. The procedures for collecting them need to be clearly specified. The resulting data need to be kept in an accessible database.

Then, to use these four core measures for estimating, bidding, and project control, the industry has to establish the *relationship* between them. Let us back up a minute and consider the relationship behind any means of doing work:

Work *Product* (at a *Quality* level) = *Effort* over a *Time* interval at a *Productivity* level

One new term, productivity, has appeared. It appears that work measurement needs a fifth core measure, productivity. Where is this measurement to come from? Let us restate the foregoing relationship in software terms:

Size (at implied Quality) = Effort x Time x Productivity

From this expression, we see (rearranging it by algebraic methods) that Productivity is a function of size, effort, and time:

Productivity = Size / (Effort x Time)

In other words, software productivity comes from the core measures, and we can acquire them from completed projects. However, software productivity has been conventionally defined as size/effort (conventionally from economic theory, output/input). Hence, we need a new name for this version of software productivity that includes the *time* schedule. We called it *process productivity*. It is the productivity, not just of an individual programmer producing code (software lines of code/person-month), but of a project full of all kinds of software people operating over the time period of the project. Hence, this version of productivity includes the capability of requirements gatherers, analysts, designers, implementers, and testers. It covers the effectiveness of management, process, modeling, and tools. And not so incidentally, it is affected by the understanding with which stakeholders approach the software task.

Unfortunately (for the sake of progress in software estimating, control, and management) people have had some difficulty grasping the fact that the schedule planned at the beginning of a project does have an effect on the productivity that the software process can achieve.

To define this relationship more precisely, Putnam analyzed a broad database of completed projects. The general *work* relationship set forth above held, but the resulting equation turned out to be nonlinear, that is, effort and time were raised to powers. The basic software equation would then be:

Size (at some specified or attainable Quality)=Effort^a x Time^b x Process Productivity

The databases from which Putnam derived the relationship supply the values of the exponents, *a* and *b*, but those databases do not supply the value of process productivity at which the software organization will carry out its next project. This value is best calibrated from immediate past projects. Thus, process productivity becomes the *fifth* core measure, derived from three base measures.

The other input value to the estimating relationship – size – is appraised from what is known about the project at the time the estimate is prepared. If the size estimate must be made before much is known about the product, it will likely vary substantially from the eventual completed size. The corresponding effort and time estimates may, accordingly, be equally far from the eventual reality.

In software development, the relationship between effort and time is multiplicative. That is, if the size and process productivity terms are considered to be fixed in a particular application, estimators can shorten time only by increasing effort. Or, if they wish to reduce effort (and cost) they must allow more time. According to this relationship, effort and time must move in opposite directions.

Stakeholders Must Use the Relationship

Measurement systems may be operated in a detailed sense by a specialized measurement staff or even by project management. *Details* refers to the definition, collection, and databasing of the measures and the mechanics of using them for estimation and control. In addition, all the stakeholders from client management onward must understand metrics-based

management, in a broader sense.

For instance, it is often high-level management that sets the finish date to meet business commitments and lower levels that work out the requirements. Neither group may understand clearly that the amount of work needed to satisfy the requirements determines the effort. Moreover, they often fail to understand that the time (schedule) needed depends, as the foregoing relationship demonstrates, on the amount of effort financed. Furthermore, stakeholders have a strong hand in other factors entering into project estimating and bidding, such as the following:

1. A key measure in the estimating relationship is the size of the eventual product. Size is often measured in source lines of code, but it may be represented in any unit that represents the functionality to be produced by the project. Selection of that unit is a management chore, often involving the client as well.
2. The amount of functionality is clearly unknown when the eventual product is little more than a gleam in some high executive's eye. Estimation accuracy increases if it can be deferred until the product is delimited, defined, and de-risked. De-risked. Now there's a word you won't find in your dictionary. Books abound on software risk management. All we mean at this point is that you have to identify the important risks and mitigate them. That does not mean you have to solve them before bidding. It does mean you have to foresee an approach to solving them – an approach for which you can allow sufficient time and effort in the estimate and bid. Clients and management control how long making this firm estimate can be deferred, not the project or estimating staff.
3. Basically, there is a trade-off between effort and time. Within limits, planners can reduce time by increasing effort. They can reduce costs (effort) by allowing a little more time. They cannot have both at the same time, except by increasing process productivity and that takes time on a longer scale than project schedules. Therefore, stakeholders must be satisfied with a reasonable trade-off. There is no golden shortcut. That is a hard lesson for clients and higher management to recognize and accept.
4. Clients and management are up against business imperatives – get it done, in a short time, at an effort (cost) within the client's reach. But development is up against another set of imperatives symbolized by the relationship set forth above. The two sets of imperatives have to be reconciled. It helps if clients and executives have a grasp of the software relationship. It also helps if software managers understand business pressures. Sometimes they can strip down project functionality to come closer to business schedule-and-effort goals.

Finally, the software relationship is not a law of physics, that is, each term is uncertain to some degree. We may gain some insight by comparing the software process to a communication channel, as researcher Claude Shannon employed the concept [1]. Shannon originated the mathematical theory of transmitting information over a communication channel such as a signal over a wire.

1. The channel had a certain capacity or bandwidth – a transfer rate in bits per second.
2. It had a certain amount of noise, random electrical signals arising out of the environment that interfered with the transmission of the bits carrying the information.
3. As a result of capacity limitations and noise, some of the bits carrying information were distorted in transmission. That was an error rate.
4. This error rate may be reduced by improving the channel or adding error-correction algorithms.

Similarly, we may conceive of software development as taking place through a channel called a process, extending from systems definition and requirements capture to delivery.

1. This process has a certain capacity – mechanically measurable as bits at the input to the digital device that uses the software.
2. It generates a certain amount of noise, or defects, resulting from deficiencies in the process or errors by the people engaged in the process.
3. As a result of these deficiencies and errors, some of the output bit-stream is incorrect.
4. This defect rate may be reduced by improving the process (for example, instituting reviews) or correcting the product (for example, testing).

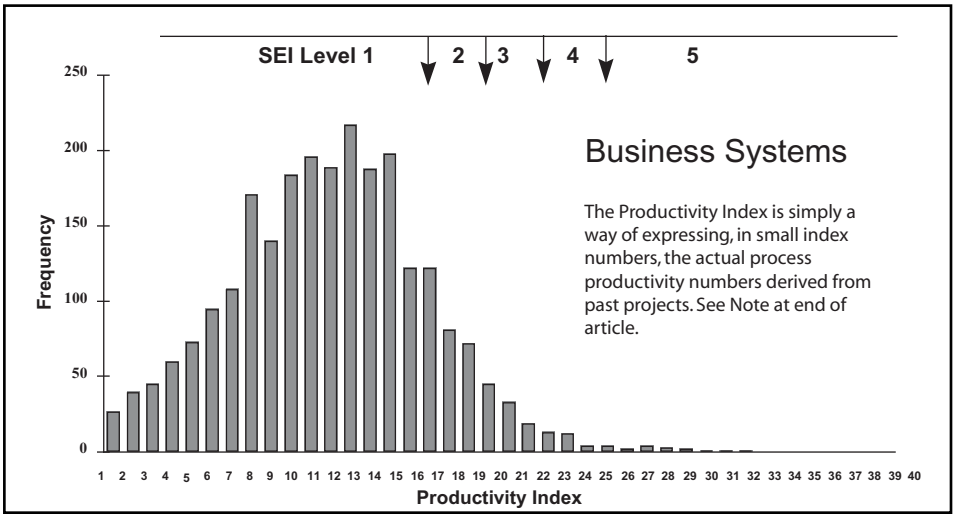
In this theory, the signal containing the information gets through the channel (usually – sometimes it is overcome by noise). However, the signal is normally afflicted with noise, and the receiver has to pick the signal out of the noise. In other words, a communication process is *uncertain*.

Similarly in software development, the product gets through the requirements-analysis-design-implementation-testing channel (maybe two thirds of the time), but the numbers that measure its progress through this channel are cloaked in uncertainty.

For example, the size is uncertain until the product is shipped. The process productivity is uncertain because the next project may have different staff, run into different problems, and so on. Consequently, the effort and time derived from the software relationship are also uncertain. Sometimes they are so uncertain that the project fails to complete. It is overcome by noise. It runs out of time and effort and has to be replanned.

By accepting this data uncertainty (or noise), planners can employ the principles of probability to provide enough time and effort to increase the chance of successful

Figure 1: *Quantitative Software Management – SEI Mapping: Example is for the process productivity of QSM's 5000-system database, distributed over a long range*



completion to whatever level management sets. For example, it could ask for 50 percent. At this point the odds are 50-50 that the project will complete within the expected effort and time estimates. It could set 80 percent, providing an increased assurance level by including time and effort buffers beyond the expected time and effort.

Process Productivity Proves its Worth

The Software Engineering Institute (SEI) at Carnegie Mellon University worked out the five-level Capability Maturity Model® (CMM®) in the late 1980s. Since then hundreds of software organizations have advanced from Level 1 to Levels 2 and 3. A few have made it to Levels 4 and 5. As a matter of observation and common sense, the effectiveness of these organizations has improved as they gained levels. Or, in the terms of the relationship set forth above [Productivity = Size / (Effort x Time)] their *process productivity* has gained.

Companies can compute process productivity from three of the core measures of completed projects. That makes it an objective number, not a matter of opinion. In measurement terms, it is an overall metric indicator of the effectiveness of software development of a project, a series of projects, or the development organization doing these projects. It is also a number that can be correlated with the SEI CMM levels and the work that Putnam did in the mid-90s, as brought up to date in Figure 1.

Note that the frequencies (vertical axis) conform roughly to the standard normal curve. Since many activities involving human measurements follow this curve, that fact suggests that the metric indicator, process productivity, is on the right track.

From here it is a simple mathematical step to express that increase in process productivity (with which not everyone is familiar) in terms of schedule, effort, and reliability improvement (which everyone understands). The result is presented in Table 1.

To manage software development intelligently, project managers need to understand the core measures and the mathematical relationship between them. They can then intelligently estimate, bid, monitor, control, and improve their ability to develop software over a period of time.

By basing the control of software development on these five core measures, management has the means to plan time and effort commensurate with the functionality (size) expected. By applying statistical methods to the acknowledged uncertainties of these measures, management can improve the odds of completing within a plan. By advancing through the CMM levels, management can greatly improve its core measures. The stakeholders need this understanding as well; it enables them to interact intelligently with the project managers and developers.

By extending an understanding of metrics-based management [2] to project managers, higher management, and stakeholders, the software industry can get the better of the beast.◆

SEI Level	Effort (Cost) Person-months	Peak Staff People	Schedule Months	Mean Time to Defect Days	Defects Remaining at delivery # of Defects
1. Initial	1542	91	26.0	0.34	255
2. Repeatable	831	60	21.2	0.52	137
3. Defined	241	26	14.0	1.18	40
4. Managed	92	14	10.2	1.78	21
5. Optimized	38	8	7.6	4.06	6

Table 1: *Increase in Process Productivity Expressed in Schedule, Effort, and Reliability. Example is for a 100,000-source-lines-of-code (SLOC) engineering system*

Further Information

More information on the five core measures [2] is available at: <www.qsm.com>.

References

1. Shannon, Claude E. "The Mathematical Theory of Communication," Bell System Technical Journal 27 (1948): 279-423, 623-656. Also University of Illinois Press, 1949.
2. Putman, Lawrence H., and Ware Myers. The Intelligence Behind Successful Software Management. New York, N.Y.: Dorset House Publishing, 2002 (to be published in the fall).

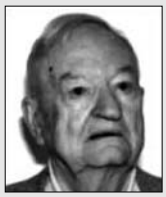
Note

The actual values are very large, ranging from 754 for Productivity Index 1 to 1,346,269 for Productivity Index 32, the highest shown on the figure. Each Process Productivity value is 1.27 times the previous one. The details depend upon the particular units of measurements used such as years for time and person-years for effort. We are trying to explain the general idea very briefly, not treat all the details.

About the Authors

Lawrence H. Putnam is president of Quantitative Software Management, which he founded in 1978. Previously, he spent 25 years on active duty, including tours in the Office of the Director of Management Information Systems, and the Assistant Secretary of the Army, Financial Management where he gained experience in software development from a top management perspective. Putnam is a graduate of the U.S. Military Academy at West Point.

Quantitative Software Management,Inc.
2000 Corporate Ridge, Suite 900
McLean,VA 22102
Phone: (703) 790-0055
Fax: (703) 749-3795
E-mail: larry_putnam_sr@qsm.com



Ware Myers is a contributing editor to *Computer*. Myers assisted Putman in 1981 with his first tutorial book for the Institute of Electrical and Electronics Engineers Computer Society. This was the beginning of a long writing collaboration. Myers is a graduate of Case Institute of Technology and has a master's degree from the University of Southern California.

1271 North College Avenue
Claremont, CA 91711
Phone: (909) 621-7082
Fax: (909) 948-8613
E-mail: myersware@cs.com