What We Have Learned

Lawrence H. Putnam

Ware Myers

Quantitative Software Management, Inc.

What have we learned about the application of metrics to software development in the last quarter of a century--a long stretch by the standards of the software profession? For one thing, we could not help noticing that an almost endless string of fads has come--and in a little while, gone. The one thing that is solid in all this change has been actual measurements. And among scores of measurements, the most solid have been the SEI core metrics: functionality (usually expressed as a measure of size), schedule time, effort (convertible to cost), productivity, and defect rate. They are solidly based because most software organizations have something like them. Consequently, when we began to look 25 years ago for enough data to draw some conclusions with which to guide operations, they were what we found.

Lesson No. 1. Conventional productivity is deceptive

Productivity in most manufacturing operations is *linear*, that is, when a cobbler worked longer hours, he produced more shoes. When he hired an assistant, he soon doubled production. Quite naturally, this frame of mind carried over into software development. Software managers adopted the linear expression, source lines of code per person-month, as their gauge of productivity.

Is it the correct frame of mind? No, it is not. When we examined the actual data, source lines of code and person-months, on thousands of completed projects, we found two facts to the contrary:

- It takes more effort per source line of code to build larger systems than it does smaller systems. Another way of putting this fact is to say: if effort per SLOC were the same at all system sizes, the relationship would be *linear*. Since effort per SLOC is not constant at all sizes, the relationship is *nonlinear*.
- Moreover, at any one system size the effort required to build it differs widely. For example, in the project data that we have collected, at a size of 100,000 SLOC, effort ranges from a low of 10 person-months to a high of 2500 person-months. The extent of the effort is equally great at other system sizes.

These two facts should not be unexpected. Large systems generally are more complex than small systems. The relationships between their parts are more intricate. Larger teams of developers complicate their interactions, as compared to small teams, and so on.

Similarly, with respect to the second fact, there are great differences in system complexity between developments in different application areas. Experience tells us, for instance, that real-time systems take far more work than business systems of the same size in lines of code.

This nonlinearity of the relationship between SLOC and effort means that the conventional productivity that a software organization derives from its experience on one size or application type is not a reliable guide to estimating the next system of different size or application type.

Let's bring in a second core variable, *time.* Again, we see the same nonlinearity.

- Development time per SLOC increases with system size,
- Development time varies widely at each system size. For instance, at about 10,000 SLOC, development time ranges from about two months to 80 months.]]

We learned that measuring software productivity in the conventional way, SLOC/person-month, is deceptive. Software productivity is more complicated than that:

- Productivity is affected, not only by effort, but also by *time.*
- The relationship of the three metrics--size, effort, time--is *nonlinear.*

The true relationship is a *nonlinear* version of:

$$\text{SLOC}/ (\text{person-months}^{a} \times \text{calendar months}^{b})$$

The exponents, *a* and *b*, represent nonlinearity. We call the result, process productivity.

Lesson No. 2. There is a minimum development time

There is always pressure on development organizations to get the system out faster. That is quite understandable. We live in a competitive society. Faster is better. As we examined the recorded metrics of hundreds of systems over these 25 years, however, we came upon an interesting reality.

- No one had ever completed a system in less than a certain *minimum development time.*

We came upon this fact in the course of the studies that led to Lesson No. 1. There were no data points below a certain value of schedule time and, we might add, they were mighty sparse just above that value. There are no data points for a simple reason: No one has been able to do a system in that short a time.

Of course, one has to define what the "system" is. It is a specified collection of requirements or features at a quality level. If you sacrifice some of the features, you have less work and you can reduce the minimum time in proportion. If you sacrifice quality,

for instance, by abbreviating system test, you can ship a little sooner. To complete the system originally intended, however, takes at a minimum a certain amount of time. You can't beat that minimum time just by adding more people.

Knowing what the minimum development time of the prescribed system is, based upon hard metrics, can be a great comfort when you are standing before the decision-makers. Contrariwise, when the decision-makers are contemplating a set of bids, they can use that same knowledge to throw out the bidders who don't understand metrics. The "low" bidder is not always a bargain!

Lesson No. 3. You can trade off time and effort

The minimum development time sets a lower limit. At this limit, effort (or staff or cost), and defects are at a maximum. You may not be wildly happy about operating at minimum time. After all, cost and quality are important values, too. By extending development time, you can reduce effort and defects. The upper time limit is a matter of judgment, but it is around 130 percent of the minimum time. Beyond that point further reductions in effort and defects are small. Within that time range, you can balance time, effort, and defects to fit your business pressures.

Lesson No. 3A. Small is better

Other things being equal, a small staff is more efficient than a large staff. It is more efficient because there are fewer interfaces between project members. Staff spend less time communicating, but yet keep better informed.

Our metric observations have established the reality of "small is better" all along, but a couple of years ago we obtained data on 491 medium-sized projects. As Figure 1 shows, the two-to-five person teams completed projects of comparable size with about one-third of the effort of the seven-to-14 person teams.

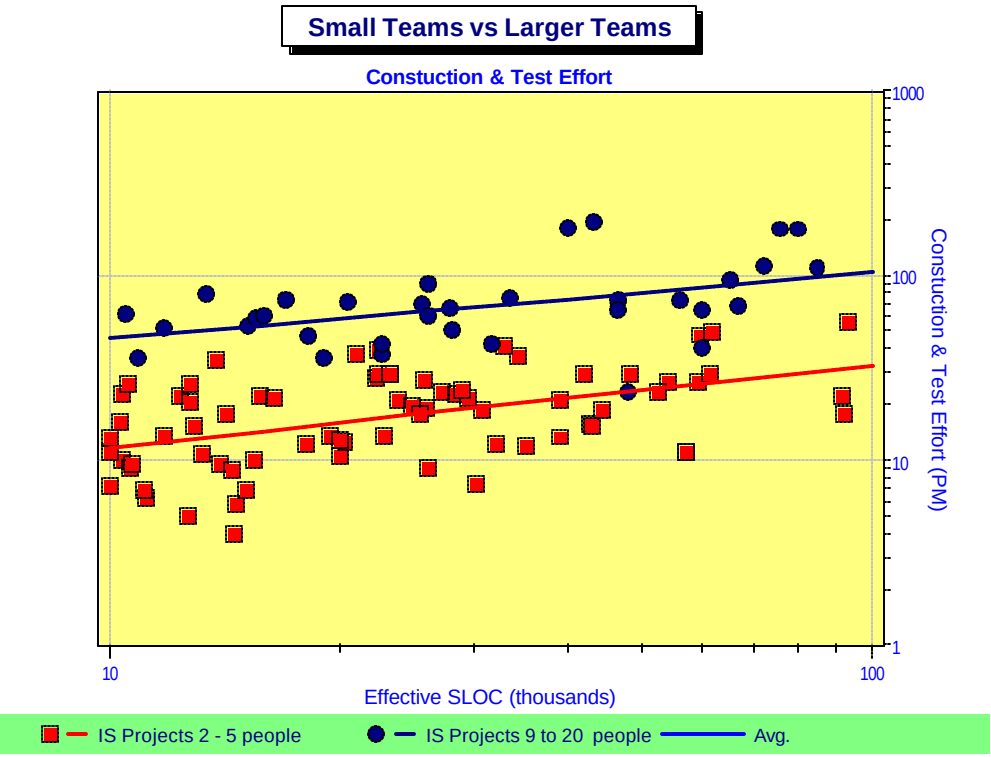

Figure 1. The larger teams (upper line) take much more effort than the smaller teams (lower line) at each system size.

Lesson No. 3B. Fewer defects are better

There is also a time-quality tradeoff. When you extend your development time beyond the minimum, you score fewer defects. Sophisticated users are noticing that the first product to market generally tries their patience.

Lesson No. 4. You can live with uncertainty

[[When we collected data on hundreds of projects--now over 5,000, the uncertainties in any one core metric were balanced by offsetting uncertainties in other metrics. As an individual, on the contrary, you are dealing with one project at a time. Yet the data you must work with is uncertain. For example, in estimating time and effort, you start with values for system size and your project team's process productivity. At the time you have to bid, both the system size and the process productivity of the team that you will assemble for that project are uncertain. By *uncertain* we mean that you cannot pinpoint the value. You estimate the size, for instance, to be 77,000 SLOC, plus or minus 11,500 SLOC.

The method of statistical simulation, summarized in the sidebar, enables you to cope with these uncertainties. Briefly, this method enables you to translate the uncertain values of your input values into the probability that you can successfully complete your project at

the values of schedule and effort that you select. You have to select values, of course, within the range of what is possible.

Without the benefit of this kind of analysis, management is often tempted to go with the low bid. Granted, it is important to win contracts. It is also important to complete projects successfully and make a little money while doing so. It is also important to stay in business. This type of analysis gives you the opportunity to gauge your chances.

Sidebar: Deal with uncertainty statistically

Statistical simulation, summarized in Figure 2, provides the means of dealing with the uncertainties of your input values.

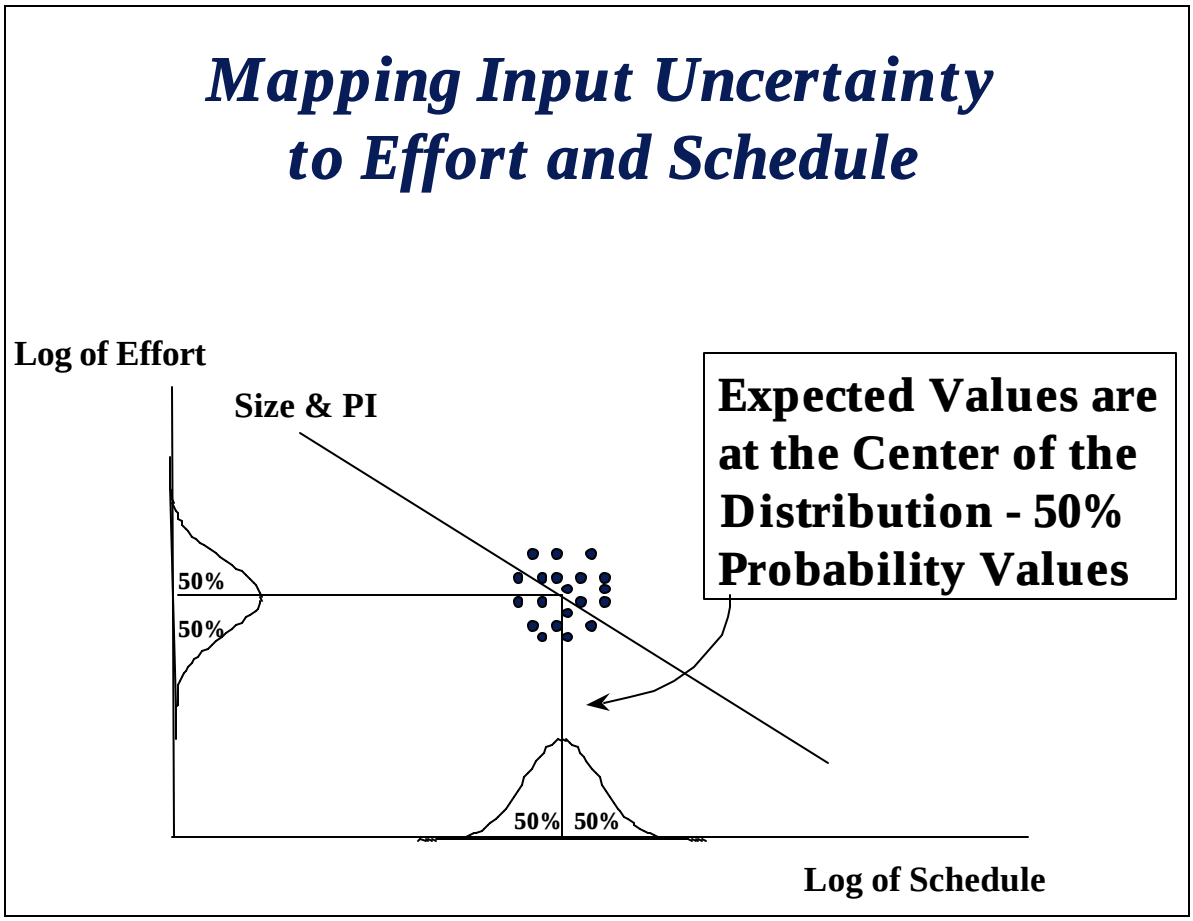

Figure 2. Computing your project estimate a thousand times provides a thousand possible answers, symbolized by the black dots. Projecting the thousand data points to the two axes enables you to find the probability that any one answer will work out.

Looking at Figure 2 in terms of seven successive concepts makes it easier to understand. We have to admit that the figure can be a bit awe-inspiring at first glance. So don't glance at it. Study it one concept at a time.

1. The diagonal line labeled “Size and PI” (meaning process productivity) symbolizes the location of your possible operating points somewhere between the minimum development time (a bit to the left of the circle of data points) and 130 percent of it (to the right of the data points).
2. Let's say the center of the circle of dots is the operating point you have chosen, that is, a particular length of schedule and amount of effort.
3. You then have a statistical program on your computer calculate schedule and effort a thousand times, leading to a thousand values of effort and schedule, symbolized by the circle of dots
4. For each computation the statistical program selects input values from a statistical distribution range around the values of size and process productivity you are using. Since the input values of size and process productivity vary, the output values of time and effort also vary. This variation is symbolized by the distribution of dots in the output circle. If we were to show a thousand output data points, we would find that they follow a probability distribution, heavy toward the center of the circle of dots, light toward the circumference.
5. Next we project the data points to the two axes. The most likely values of effort and time are the ones at the peak of each curve. In terms of probability, the chance is 50 percent that the project can be accomplished at those values.
6. Selection of a longer schedule (to the right on the schedule curve) increases the probability of project success, while at the same time reducing the effort needed--within the limits that the two curves provide, of course.
7. Selection of a shorter schedule, on the other hand, reduces the probability of project success, while increasing the effort needed.

End sidebar

Lesson No. 5. You can control a project under way

Our study of completed projects reveals that key ongoing metrics, such as the amount of effort, the number of staff, functions completed, or defects detected, fall along a Rayleigh curve, as diagrammed in Figure 3. Great! All you have to do is figure out a way to project the likely occurrence of each of these variables. (Hint, the methods are available.) Moreover, since these methods are statistical, you can also project control bands, also shown on the figure.

That provides the control side for *statistical control*. The other side is to plot actuals week by week on the control diagram. If the actuals are falling within the inner control band, work is progressing as expected. If the actuals begin to veer out of the control band, something has gone wrong. You have early notice of trouble.

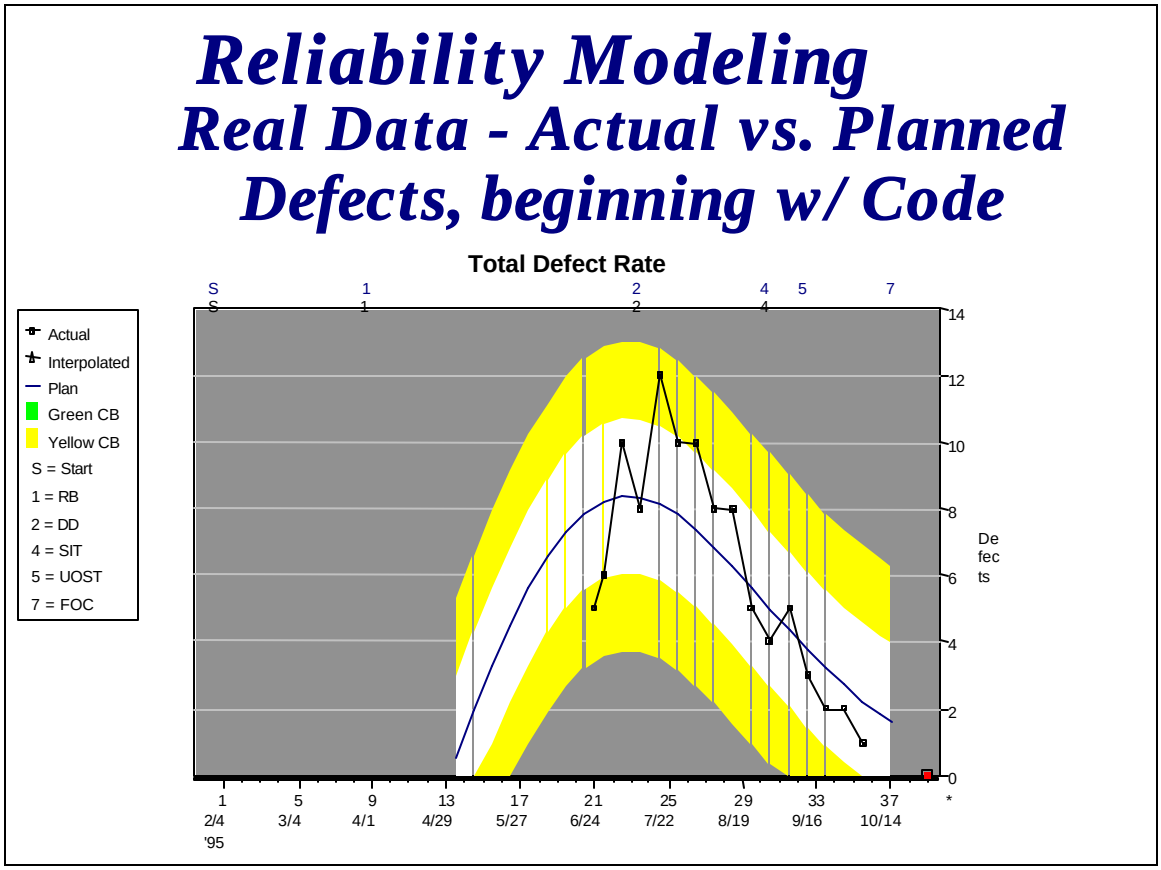

Figure 3. The solid curve is the projected defect rate. The little squares are the number of defects discovered each week. The band above and below the solid curve represents the allowable tolerance around the defect-rate curve. In week 24 the number of defects peaked, but the project manager got right on it!

Before we leave this lesson, let us take note of a frequent exception to the Rayleigh pattern. It is staffing. If management staffed a project in accordance with the needs of the project, staff would build up over a period of time as the small initial staff sorted out tasks to occupy more and more people. At some point as tasks in work reached a peak, so would staff. As the project wound down, so would staff.

Rayleigh staffing, however, takes continuous management attention, moving people from declining projects to growing ones. Management often prefers level staffing--assigning full staff at project initiation and maintaining it until completion. In the all-too-common disaster scenario where risks and defects are left until integration or system test, full staff and more are needed right up to the release bell. So, to the extent that considerations other than task needs dictate the staffing pattern, it may not follow the Rayleigh curve.

Lesson No. 6. You can replan a project midway

In spite of the best-laid plans of mice and men, projects often *gang agley*, as Robert Burns once put it. Moreover, in software development, the plans may not be laid all that well. Whatever! If we are accumulating metrics as we go along, we can use them to project a new schedule and effort to completion.

The new plan is likely to extend the schedule and perhaps to call for more staff than the budget can afford. In that event, you can cut features to meet the schedule required or the budget available. Metrics, as such, cannot tell you what to do. It can provide you with the informational means to do what your circumstances require of you.

Lesson No. 7. You can monitor process improvement

With the introduction of the Unified Process about a year ago, many organizations have more management attention focused on process improvement than ever before. Management attention is a scarce resource (managers have a score of urgent matters clamoring for attention). It helps to focus their attention by having a metric cross the desk periodically. In the realm of process improvement, such a metric is process productivity.

We derive process productivity by formula from the size, time, and effort metrics of each project. Hence it is a definite number. In an organization with many projects, it is also a frequent number. Its original use was for project estimation, but you can use it to pursue other goals as well:

- If the average process productivity of projects completed this year exceeds that of last year, you are making progress. Moreover, the difference between the two numbers indicates your rate of progress. You can be satisfied--or dissatisfied, and take action.
- The average process productivity of the organizations reporting data to our database has been improving for the last two decades. You can project a comparable improvement rate and plan the actions it will take to achieve it.
- You can compare your level of process productivity with other divisions within the same corporate structure or with industry-wide averages that we maintain.
- You can evaluate the process productivity level of subcontractors on whom you have some leverage, such as a million-dollar contract to bestow.

One use of process productivity we urge you not to employ. That is, to evaluate a particular project, its project manager, or its staff for various personnel-type purposes. Software metrics are too important for all these broader management purposes to risk muddying them up to assess individuals.

Lesson No. 8. You can profit from experience

No one individual has more usable experience than he or she can cram into that famous "little black book." That often is as little as two or three projects. Work done long ago in a "chaotic" process on a "prehistoric" operating environment may no longer be highly

relevant. Yet the agency or corporation in which you labor may have completed scores of projects in the last few years--if only you could get your hands on that experience.

Obviously, you don't have time to dig through the written records in hundreds of file drawers, perhaps scattered around the globe, perhaps in dusty dead-record warehouses. The answer in the personal-computer age is a metrics database accessible from your desk or laptop.

The general answer, the metrics database, is easy to give. Making it work is a little more difficult. Your organization has to decide what metrics to store, but start with size, time, effort, and defects--the SEI core metrics. It has to define those metrics so that they mean pretty much the same thing from project to project and from location to location. It has to institute the discipline that it takes to collect data from not-too-enthusiastic sources, at least in the beginning. In time, everyone will glory in the ready availability of good metrics.

In other words, we have "learned" that metrics have day-to-day value only if they can be readily accessed.

Lesson No. 9. You can "master plan"

There is a level of activity above the project. With our strong tendency to focus on the problems of the project, we overlook the fact that most organizations have a number of projects under way at the same time. They are not all at the same process phase. Their need for resources differs.

If you have common metrics, if the metrics for each project are in your computer, you can "master plan" the allocation of resources over time to each project. You can prioritize your projects to match your resources. Or, with the advance notice that the master plan provides, you can build up your resources to meet coming needs. Or, in reverse, if the master plan forecasts a lull six months hence, you can slow down hiring or bid new work a bit more aggressively.

As component-based development gets more play, the source of these components comes into focus. That source, wherever it is, is outside the usual project emphasis. You may have a supra-project group developing common architecture, standardizing interfaces, and developing components for a whole range of projects. You may be obtaining components from vendors. You may be charging one of your own projects with generalizing a component it needs for its own application for broader use. You need metrics on the master-plan level to facilitate component-based development.

Wrapping it all up

Underlying these nine *lessons* is the "software equation." We won't go into its details here. What we learned from it is that schedule time and effort-cost play a simultaneous role in software development. The two are irretrievably linked. You can't have one

without the other! Trying to play them separately is what has led to a lot of the trouble that has beset the field.

The "software equation" is also at the heart of the idea of *calibration.* To know what you have to do to get to where you want to go, you first have to know where you are. That is what we mean by calibration. The software equation provides that means.

For example, from the software equation we derive process productivity. Knowing how productive our process is, is a key ingredient of planning and estimating. Then, during a project we can find out if we are getting the process productivity we originally estimated. If we are, good. If we are getting something less, we can replan the remaining work before it is too late. Finally, process productivity provides the basis for evaluating productivity between projects and over time. It is the beacon light for your process improvement activity. And process improvement is what keeps you in business in these tumultuous times.

Further information

The Web-site, qsm.com, lists about 40 articles, explaining the lessons of this article at greater length. In addition, Putnam and Myers are authors of three books:

Measures for Excellence: Reliable Software on Time, Within Budget, Prentice-Hall, 1992

Industrial Strength Software: Effective Management Using Measurement, IEEE Computer Society Press, 1997

Executive Briefing: Controlling Software Development, IEEE Computer Society Press, 1996.

Author biographies:

A graduate of West Point, Lawrence H. Putnam spent 25 years on active duty, including tours in the Office of the Director of Management Information Systems and the Assistant Secretary. There he viewed the problems of software development from a top-management perspective. In 1978 he founded Quantitative Software Management, Inc. in McLean, VA, and continues today as its president. In QSM he brought his concepts of software estimation and control, originated in the Army, to full fruition.

A graduate of Case Institute of Technology with a master's degree from the University of Southern California, Ware Myers, as a contributing editor of Computer magazine, was impressed by one of Larry's first public presentations of his ideas at Compcon 77. Larry helped Ware with a long article in the magazine. Then in 1981 Ware helped Larry with his first tutorial book for the IEEE Computer Society. And they have lived happily ever after, with article after article, and book after book.

Contact information:

Lawrence H. Putnam, Sr
Quantitative Software Management, Inc.
2000 Corporate Ridge, Suite 900
McLean, VA 22102
Tel 703 790 0055
Fax 703 749 3795
Email: Larry_Putnam_Sr@qsm.com
www.qsm.com

Ware Myers
1271 North College Ave.
Claremont, CA 91711
909 621 7082
myersware@cs.com
Fax: 909 948 8613

Abstract

In all the changes in the software field in the last quarter century, one solid element has been the SEI core metrics: functionality (usually expressed as a measure of size), schedule time, effort (convertible to cost), and defect rate. From these four metrics, we derive a fifth, productivity. These five are related to each other. From these relationships, we derive a number of *lessons*, which it is the business of this article to identify.