Familiar Metric Management: What Will the Year 2000 Fix Cost Me?

Estimates by consulting groups of the worldwide cost to fix the Year 2000 problem range from $300 billion to $1200 billion (not counting the cost of litigation and legal damages that may result from failure to fix). In contrast, organizations that actually face the problem typically estimate much less. For example, the US Federal government has budgeted only $2.3 billion over the next two and a half years. Estimates by consulting groups for the Department of Defense alone are in the $30 billion range.

In a front page article April 7, the New York Times featured the problems the Mason Shoe Company, Chippewa Falls, Wisconsin, is facing, summarized in Table 1. Mason Shoe's president, John Lub, agreed to this $500,000 Y2K budget reluctantly. "We get absolutely nothing out of it," he told the New York Times. "We get to stay in business, that's all." Analysts believe some companies may exit the scene entirely for failure to overcome the Y2K problem.

Table 1. Mason Shoe Company is working on the Y2K problem with a $500,000 budget.

Employees	750
Lines of Code	3 million
Programmers on the Job	2
Consultant, two days per month	$1200 per day plus expenses

We need an estimating "fix"

In an effort to get a "fix" on the financial magnitude of the Y2K problem, we ran a simulation for a typical medium-size company with the following numbers:

200 systems: inventory of existing systems to be modified, excluding systems that are no longer needed and systems for which replacements are being purchased.

18.4 million SLOC: the size distribution of these systems is typical of that found within QSM's database of more than 3000 business systems.

92,000 SLOC: average system size.

6 percent, or 1,104,000 SLOC: estimated number of new and modified lines of code to be worked on.

$14,583/person-month: fully burdened labor rate.

Process Productivity Capability of fixing organization: average.

We ran these numbers through the algorithms in the SLIM estimating tool 1000 times. The results of this simulation are summarized in Table 2. We assumed that the team sizes were kept small, two or three people, typical of the number to be assigned to fix an individual system among the 200 in the inventory. We included all the Y2K work: planning, design of the process, finding and fixing the problems, and testing the result.

The employment of simulation methods, that is, running 1000 trials, captures the non-linearities in the software development process. Practitioners have noted this non-linear tendency over the years. Simulation methods also provide a realistic range of answers because they take the uncertainties of the input values into account. In the present case, for instance, the number of lines of code involved, the distribution of sizes, the percentage of lines that will have to be added or modified, productivity, staffing style, and burdened cost per person-month—all are uncertain. As a result of these uncertainties, the mean of the values obtained by the 1000-trial simulation is about 50 percent greater than the one based on a single computation.

Table 2. For a medium-size company the cost of fixing the Y2K problem will be in the tens of millions of dollars.

Statistic	Value
Trials	1000
Mean Cost	$21,760,537
Median Cost	$20,156,881
Standard Deviation	$8,228,242
Coefficient of Variability	0.38
Range minimum	$7,186,837
Range maximum	$65,648,866
Range width	$58,462,048
Mean Standard Error	$260,200

Table 3 shows the results of the same simulation applied to inventories of different numbers of systems. The expected cost is the mean value of 1000 trials in the simulation.

Table 3. Readers may get an approximate grasp of the size of their Y2K business-system conversion problem by relating their organization to the appropriate column in this table.

No. of Systems	50	100	150	200	250
Expected Cost, $millions	$5.44	$10.88	$16.32	$21.76	$27.20
Cost Range, $millions	2 - 16	4 - 33	5 - 49	7 - 65	9 - 82

Figure 1 shows the range of an organization's costs for Year 2000 fixes. Because of uncertainties in the input values, the cost range is wide.The expected cost/SLOC is $1.09 across the whole spectrum of number of systems and spans a range from $.36 to $3.19 per line of code.

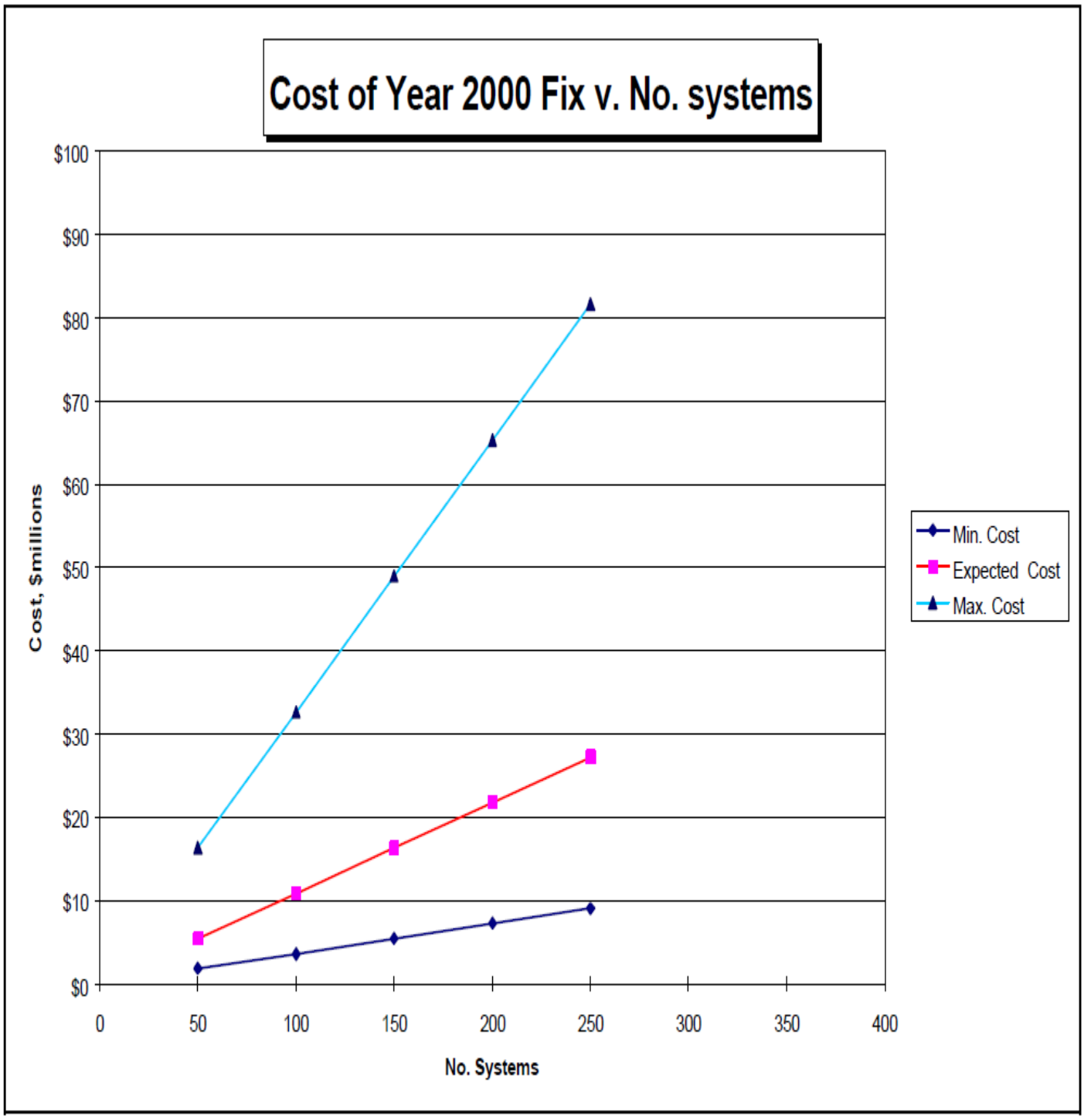

Figure 1. The middle line is the expected cost of date correction, based on the simulation. The upper line represents the maximum value. The cost is very unlikely to be less than that shown by the lower line. The lines can probably be extrapolated a little with reasonable safety.

To use Table 3, look in the column nearest to the number of systems in your own inventory. The behavior in the table is linear, so you may interpolate between columns. Of course, your systems may differ from the ones we took as typical. Your productivity may differ from the average productivity we used in the computations. Still, the table should give you a rough idea of what to expect.

More simply, the simulated cost per source line of code is about one dollar. Just multiply the number of source lines of code in each system or in your entire inventory by one dollar. If you prefer to work in function points, there are about 100 lines of Cobol per function point. So the cost to correct a function point worth of Cobol is about $100. Just multiply the number of function points by $100. These are the ball park estimates of the cost you can look forward to absorbing in the next two and one half years.

The Mason Shoe Company appears to be doing very much better than the Table 3 values. It has estimated its cost for three million lines of code at $500,000. According to the table, fixing and testing 3,000,000 SLOC would cost $3.27 million, more than six times Mason's estimate. Perhaps Mason is operating at the efficient end of our figures—36 cents per line of code. That would still cost $1.08 million, about two times Mason's estimate. It is possible, of course, that Mason is super-productive. The more likely explanation is that Mason has underestimated its costs—a flaw that seems to be endemic to the field.

What would be the cost to the Department of Defense under our formula? The New York Times quotes Emmett Paige, Jr., the Pentagon's top computer executive, as saying DoD has 9300 computer systems running 112,000 programs, not counting those at the National Security Agency, Central Intelligence Agency, and in super-secret programs. If DoD's programs average 92,000 SLOC each (a big IF), it has about 10 billion SLOC. If the cost of fixing its code is about one dollar a line (another big IF), there goes $10 billion. Paige said the current estimate of conversion costs is $1.1 billion, but he expects it to rise.

Getting ready for the fix

The main thrust of this article is on estimating, costing, and scheduling the Y2K conversion effort. The availability and value of tools for finding and fixing Y2K instances and testing the revised code is outside its scope. First you have to get ready.

Inventory your software. In many companies software has grown like a weed on Vigoro. They have no organized record of what they have. The first step is to inventory the existing software and to assign it to categories:

1. The program is no longer in use. Most companies have had no organized procedure forformally retiring programs no longer needed. Now is the time to kill these programs, not to spend money making them Y2K compliant.
2. The program is no longer in routine use, but it might, for instance, be used sometime to readold 1988 files recorded in a now outmoded record format in the event some one sues the company! Mark these programs to be used only with old files. They need not be converted.
3. The program can be readily replaced with a new, purchased Y2K-compliant shrinkwrapproduct. Do so and kill the old program. Note, however, that it is now too late, in most cases, to replace all the legacy applications. Doing that is a big job.
4. Some of the existing inventory may already be Y2K compliant. Your people may have beendeveloping compliant programs the last several years. Recently purchased programs may be compliant.
5. That leaves, unfortunately, a large residue of programs that are still in use and are notcompliant.

Select pilot conversions. As you have noticed in our introductory analysis, using typical inputs results in a very broad range of output values. You can substantially narrow this range by measuring several pilot projects. Select system that are typical of your software inventory. The metrics you need are:

1. Number of SLOC in the entire pilot system;
2. Number of SLOC modified or added new in the course of converting the pilot system; theratio of these two numbers gives you the percentage that will be new or modified; 3) Time spent in making the conversion, or schedule time; 4) Effort (person-months) devoted to the project.

These are the numbers—size, schedule, effort—from which we compute process productivity, as we have recounted in earlier column [fn]Those who missed earlier columns in this newsletter may refer to our book, Measures for Excellence: Reliable Software On Time, Within Budget, Yourdon Press, Prentice Hall, 1992, 378 pp. The SLIM estimating tool computerizes these computations, but Chapters 14 and 15 explain how to do them by hand.[/fn]. This value will now be firm, based upon your own experience in this kind of work. Also the magnitude of the work to be done, based on your own value of the percentage of new and modified code, is now much firmer.

Estimate remaining conversions. Given the percentage of lines to be corrected and the process productivity with which your teams do the work, it is now a matter of mere computation to compute effort and schedule for each remaining system. Your new estimate will fall within a much narrower range, because you now have more precise inputs. Moreover, your new estimate will now be much closer to what you will actually experience, because it is based on your own pilot experience, not on the industry-wide averages on which we based our introductory computations.

A not so typical example is shown in Figure 2. It is not typical because the company’s process productivity index, based on actual performance on previous projects, is 22.2. This number is better than one standard deviation above the mean of organizations doing business systems. Also the example is not typical because the percentage of code to be modified or added is somewhat lower -- 4.2 percent. The manpower buildup index employed in the calculation is 6.8, representing a very rapid buildup of staff and a large team for this size project.

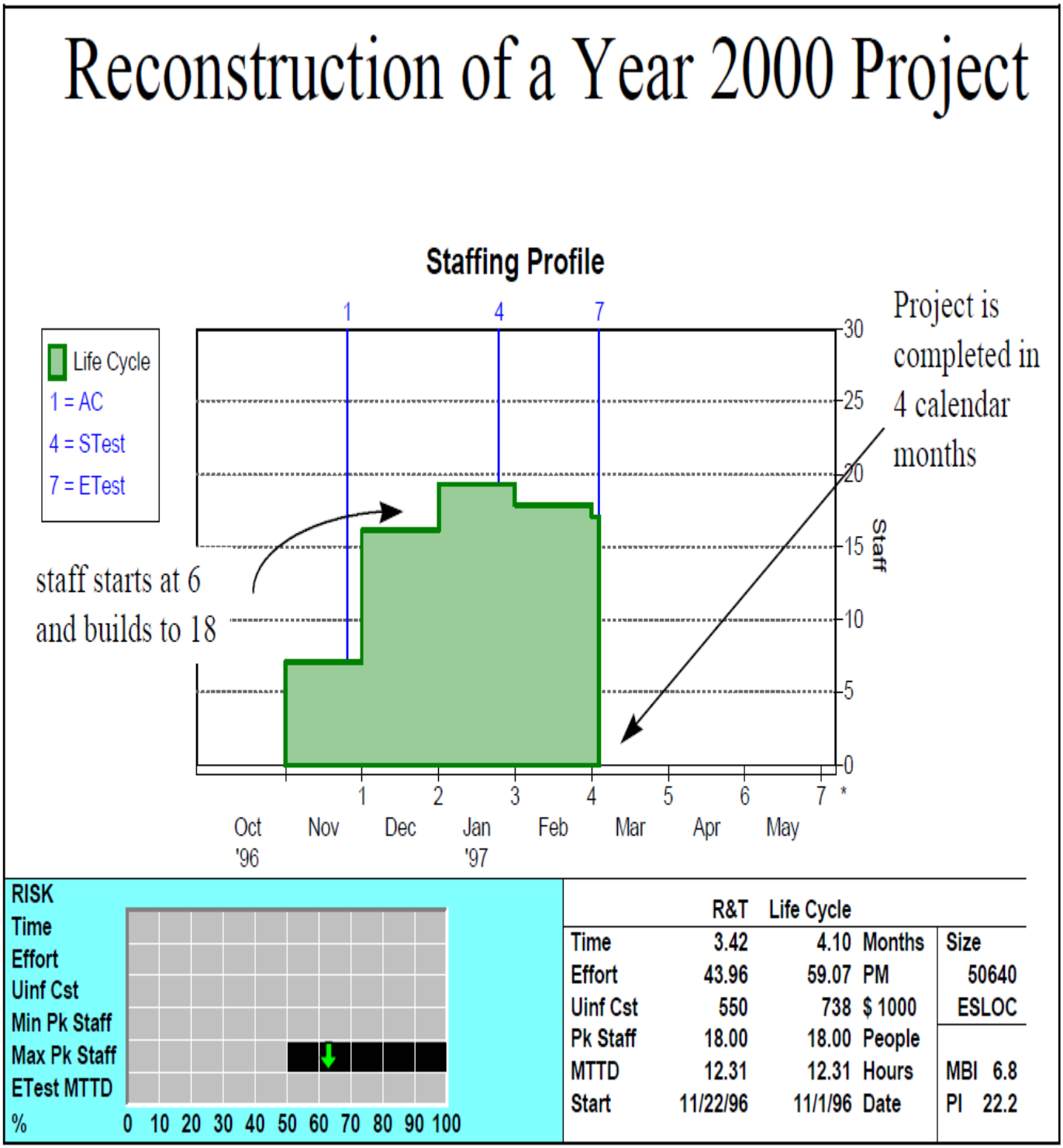

Figure 2. Given a productivity index of 22.2, manpower buildup index of 6.8, and project size of 50,640 lines of code, the staffing profile starts at six people and builds up to a peak of nearly 20 over a period of 4.1 months.

The schedule time for this project is among the best in our database, as shown on Figure 3. The cost is about average, but the productivity index is at the high end of an approximately normal distribution.

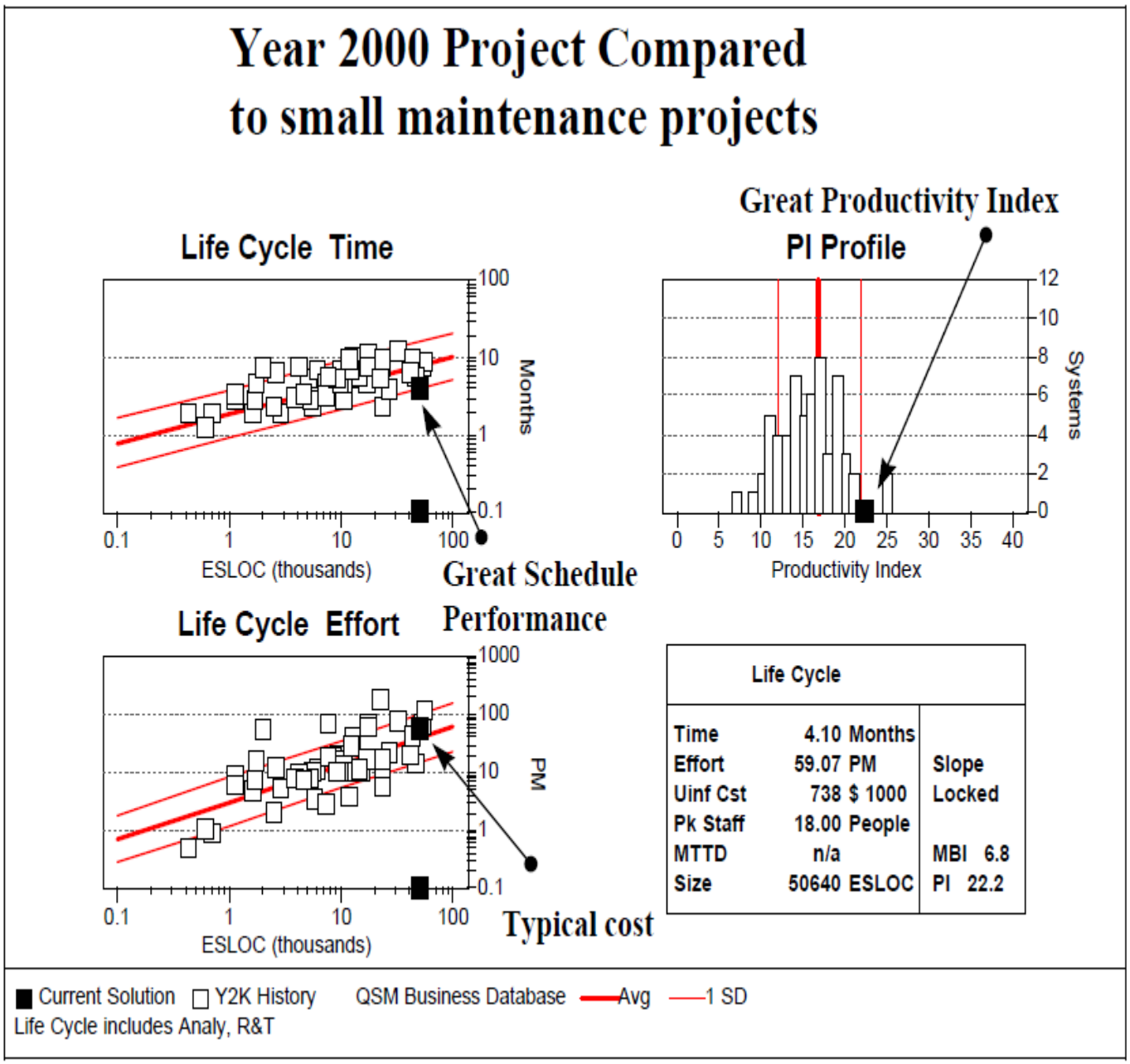

Figure 3. Schedule performance is "great" at this company's high productivity level. Because it built up staff rapidly to accomplish the project sooner, the effort (and hence cost) is "not great." It is an example of the effort-time tradeoff.

Effect of time compression

Of course, everybody is hollering "Wolf" now, even the New York Times. It is true that the period to December 31, 1999 is now extremely compressed. Some organizations have been pushing their systems into Y2K compliance since the 1980s. As we have pointed out in previous columns, there is such a thing as the "minimum development time." It is the least time in which a project of a given size has ever been completed at a given process-productivity value. Moreover, in the short run (and the two and a half years to 2000 is a short run) there

is little you can do to shorten this minimum development time. Improving process productivity reduces the minimum development time, but that, itself, takes time measured in years.

ADVICE: *Get started right away.*

You may find out, when you have completed your pilot projects and made your estimates, that the time remaining is, unfortunately, less than the minimum development time, no matter how many people you put on the conversion work. Moreover, the number of people qualified for this work is limited. Recruiters are feeling the shortage. Salaries are rising.

Triage. As all good television addicts know by now, triage is the process of sorting the severely wounded or ill into three groups, enabling a limited number of doctors to work on the most severe cases that can still be helped. Similarly, in software development it is the process of reducing the amount of work to be done in the schedule time available to what can be done. In the Y2K case, divide your remaining systems into groups:

- Work first on those that involve life and death if the dates are faulty. In your own case, work first on those that are critical to the continued operation of your organization;
- Work next on those that are significant;
- Put in the third category those where faulty dates are merely irritating, not costly. You can work on them after January 1, 2000.

The next millennium. Organizations that get a late start on Y2K conversion can confidently expect to enter the next century with some number of systems not yet fixed. Even those who have “fixed” all their systems can expect some percentage of errors to turn up when the systems actually operate after midnight December 31. For one thing, there is the error injection rate, well known to software maintenance people. It appears to be in the range of 7 to 12 percent. Y2K corrections will probably have an error injection rate in this vicinity. For another thing, software people have been delivering systems right along with only 85 to 95 percent of the defects removed.

The point is: there has been a considerable percentage of defects in operating software systems all along. The world has managed to survive. Early in the 2000s that percentage will be a little higher than it has been—or possibly quite a bit higher. Will the world survive? Or, perhaps more to the point, will your company survive? Hang around and see!