Conclusion

In software development there are many real-world problems with solutions that are facilitated by measuring the right things at the right time in the process. Process-improvement programs need to include measurement not only as the means for identifying the areas for improvement but also as the means to determine progress against improvement goals. If you are responsible for a measurement program or process-improvement program, the first thing you need to do is determine what problems need solving the most, and then start measuring.

The problems presented here are by far, not inclusive of all the challenges you are taking on with your improvement program. Chances are really good if you are just starting out with measurement and process improvement, that some of these problems are on your list. And the measurements and metrics presented here will not offer completely comprehensive solutions to these problems — they are presented as good starting points.

Once you get comfortable with the role of measurement in your processes, you will begin to identify other measurements that will help these processes and your organization mature.

About the Author

Arlene Minkiewicz is the chief scientist at Price Systems LLC. In this role, she is responsible for the research and analysis necessary to keep the Price Estimating Suite responsive to current cost trends. She has more than 15 years of experience with Price, designing and implementing cost models. Prior to her current assignment, Ms. Minkiewicz functioned as the lead of the Product Enhancement team with responsibility for the maintenance and enhancement of all the Price products. She speaks frequently on software measurement and estimating and has published articles in *Software Development* magazine and the *British Software Review*. Ms. Minkiewicz has a BS in electrical engineering from Lehigh University and an MS in computer science from Drexel University. She is a member of the International Society of Parametric Analysts and International Function Point Users Group.

How to Select Software Project Macro-Estimation Tools

by Stan Rifkin

Over the years, I have developed a checklist for selecting automated tools to perform macro software project estimates. Mine is similar, though not identical, to Robert Park's *Checklists and Criteria for Evaluating the Cost and Schedule Estimating Capabilities of Software Organizations* (SEI-95-SR-005, January 1995).

There is a zeroth item, one before selecting and evaluating an estimation tool: establish your project commitment process and then the part within that that focuses on estimation. Park, formerly at the Software Engineering Institute (SEI), has written a number of SEI technical and special reports on the subject of estimation processes, and I recommend the reader to them. The reports are available for free on SEI's Web site (www.sei.cmu.edu/publications/) and are cited at the end of this article.

Park's Fig. 5-1 (shown here in Figure 4) in *Software Cost and Schedule Estimating: A Process Improvement Initiative* (SEI-94-SR-03, May 1994) is one of the most-cited examples of an overall software macro-estimating process.

Basically, for the estimation process to support a commitment process, the following has to happen:

- Commitments have to be based on the work to be performed; therefore, there must be agreement on this.
- Estimates have to be based on (a) the work to be performed and (b) historical records of performance.
- Commitments must not exceed the capability to perform or else there is no reason to estimate.

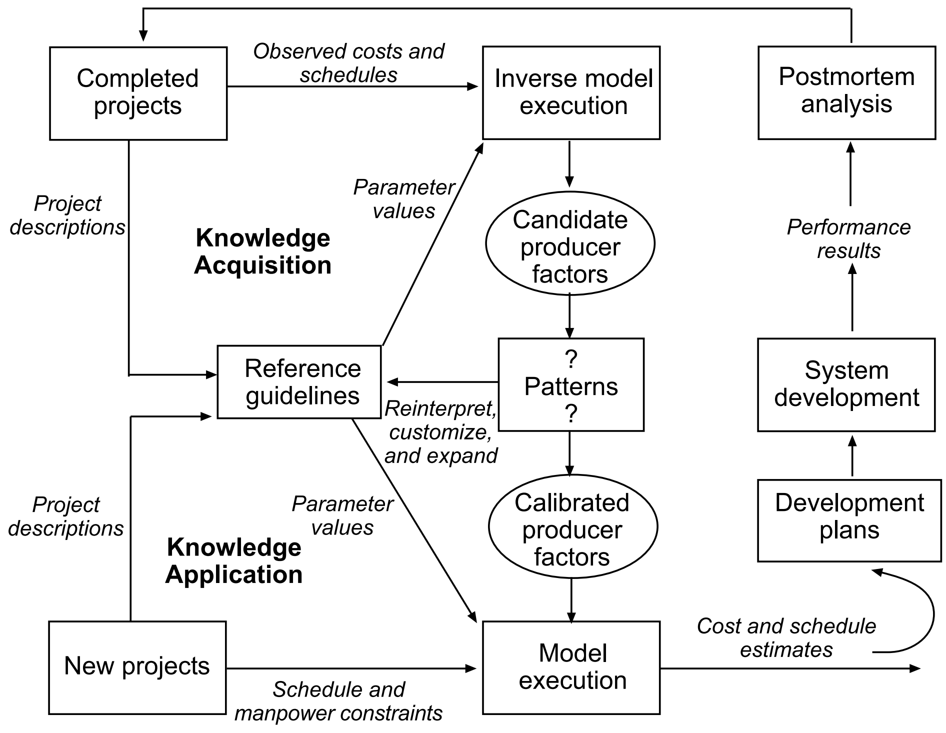

Figure 4 — Graphic template: parametric estimating. (Source: Software Engineering Institute)

Incidentally, these form the basis of the Capability Maturity Model (CMM), particularly the lower levels of software process maturity. CMM is about meeting commitments, and one of the key observations is that it is easier to meet rationally justified commitments via estimates based on reason and actual historical performance than to spend time creating stories about why we are late, over budget, etc.

With a commitment process and estimation subprocess in hand, and the admonition to select a tool that is consistent with both, here is my checklist of criteria, in priority order. The items at the top of the list dominate, so failing those top criteria can disqualify a candidate tool from further consideration.

Checklist for Tool Selection

The following items are included on my checklist of issues I look at when selecting tools.

1. The underlying algorithm is published in the public domain.

I am worried about surprises, so I want to read about the formulas used, assumptions made, constraints, etc. I am particularly looking to see if *duration = effort/resources*. It is built into every project management software package (such as Microsoft Project), and it states that the time it takes to complete a project is the person-days required divided by the number of people. So, if I have a 10-person-day project and 2 people, it would take 5 days. If I had 10 people, it would take 1 day.

If this formula is in the tool, I would have to reject it, as it is well known that the relationship among duration, effort, and resources is not linear. It may be true in building construction that doubling the number of carpenters cuts their duration in half, but doubling the number of programmers probably stops the software project!

Tools such as SLIM, Price S, and all generations of COCOMO publish their formulas, so at least I am not going to be surprised!

2. It accurately estimates completed projects.

We use a few typical completed projects to see if the candidate tool estimates them accurately in retrospect. Clearly, if the tool cannot accurately estimate our type of work, we do not want to use it on real, future projects.

The process of estimating completed projects, what Park in the diagram calls "inverse model execution," leads right to the next criterion.

3. I don't want to subjectively "guess" at the values of variables.

If the actual project performance does not equal the estimate then, naturally, I want to know where I went wrong. Some tools use scales to assess items such as team experience, by selecting a number between, say, 1 and 10. There may be text associated with each value, such as "new team," "worked together for many years," etc. I cannot measure team experience, I can only guess at its value. When actual does not equal estimate, I do not know which subjective estimates are off and by how much, so I eschew them. I only want to work with items that I can objectively measure during the conduct of a project. I would be especially disturbed by long lists of such subjective questions, as I would have no hope of tying my responses to actual project performance that I cannot measure.

4. The assumptions of the tool mirror my realities.

Estimates are application-area specific (see Chapter 1 in *Measures for Excellence: Reliable Software on Time, Within Budget* by Lawrence Putnam and Ware Myers, Prentice Hall, 1992), so we need to be sure that the candidate tool has been particularized for my application type (e.g., avionics, process control, business). It also has to be applicable to the size, programming language, lifecycle, and degree of centralization and formalization present in my projects. Often these aspects are not stated in estimation tools, so they are disqualified from consideration. One nameless tool uses a lifecycle that I do not, so I know that the estimates will not mirror my reality.

5. The tool will not generate an impossible schedule.

This was a common response to COCOMO and its progeny. Because it was a formula, it could compute a duration that was impossible in empirical terms. That is, it could compute a duration that had never before been accomplished by any team anywhere. Clearly there is a region or range of project values (duration, effort, rate of adding people, quality, and number of features) such that it is impossible to accomplish a project with that combination. (See *Measures for Excellence*, Putnam and Myers, p. 95.) I want to know that set of values so that I am sure to steer clear. It is an added bonus if the tool tells me explicitly what the impossible region is so that I don't have to spend time guessing, like playing Minesweeper!

6. The tool takes into account the effects of schedule compression.

We all know the quip about nine women making a baby in a month (this is a variant of *duration = effort/resources*). In software projects we are commonly asked to produce the minimum duration estimate. In other words, what is the effort required for the stated requirements such that the project will finish in the shortest time? Candidate tools that simply use a formula to compute the duration (such as the old COCOMO) might not take into account the nonlinear effect of schedule compression, that reducing the duration by 10% can increase the effort by 40%. Trying to reduce schedule by 20% is even worse. Michael Mah referred to this in previous issues of ***ITMS*** as the 200/20/6x rule, whereby doubling the effort (200%) results in only about a 20% schedule compression, but with a severe reliability penalty — a 6x rise in defects.

One can see, too, that the formula *duration = effort/resources* has the effect of not modeling compression, which is yet one more reason to avoid tools that use it. One of the reasons I like tools that model compression is that we are commonly given the delivery date during the commitment process, so I need to be able to ask and answer whether I can stand (that is, manage) the compression in the dictated schedule.

7. I want a range, not a point, estimate and the probability of achieving it.

We all know that estimates are most useful if they give us a range and probability, such as "There is a 70% chance of slight rain, and a 50% chance of significant rain." In order to intelligently make software project decisions we need the same thing. We need to know the probability or risk for each feasible band of estimates. This way we can adjust our input parameters in order to compute not only an answer about duration, effort, quality, and features, but also a risk level that the organization can tolerate. Without the range and risk level, it's all or nothing — not a very rational, or realistic, approach. How do we know on a point estimate what is the probability of achieving it?

Conclusion

Whether or not you use my list of criteria, be sure to list your own before you go shopping for a macro-estimation tool. Tool vendors have a technical name for those who seek tools without stated, written criteria: loose wallets!

Acknowledgements

This article is based on a presentation as moderator of the *January Maryland Society for Software Quality Roundtable 2000*, "Help! for estimation."

References

The following SEI references are often available for free download on the SEI Web site: www.sei.cmu.edu/publications/.

McAndrews, Donald. *Establishing a Software Measurement Process*. SEI-93-TR-16, July 1993.

Park, Robert. *A Manager's Checklist for Validating Software Cost and Schedule Estimates*. SEI-95-SR-004, January 1995.

Park, Robert. *Checklists and Criteria for Evaluating the Cost and Schedule Estimating Capabilities of Software Organizations.* SEI-95-SR-005, January 1995.

Park, Robert. *Goal-Driven Software Measurement — A Guidebook.* SEI-96-HBK-002, August 1996.

Park, Robert et al. *Software Cost and Schedule Estimating: A Process Improvement Initiative.* SEI-94-SR-03, May 1994.

Web Sites

www.incose.org/tools/tooltax/costest_tools.html. Provides an interesting, though slightly out of date, list of software cost estimating tools.

www.methods-tools.com/tools/frames_projmgmt.html. Offers a more up-to-date list, containing many other tools.

About the Author

Stan Rifkin is a principal with Master Systems Inc., a firm that offers advisory services related to software improvement. He worked at the Software Engineering Institute on software process improvement, the American Association for the Advancement of Science as CIO, and the National Headquarters of the American Red Cross as the director of systems development. Mr. Rifkin's previous articles have appeared in March and May 2000 issues of ***ITMS***. Mr. Rifkin can be reached at sr@Master-Systems.com.