

## The Cone of Uncertainty

In Todd Little's recent *Software* article "Schedule Estimation and Uncertainty Surrounding the Cone of Uncertainty" (May/June 2006), he expresses some concern about whether uncertainty really does decrease over time. In particular, he wonders whether estimates for work remaining are better (obviously, uncertainty's not an issue for work already done).

His results, however, might instead reflect the techniques used in his company, which, as described, appear to be the same technique used throughout the whole project—that is, "expert judgment."

The Cone of Uncertainty doesn't reduce itself, however. It's reduced by the improved estimation methods that become available as a project progresses. During the early, envisioning phase, requirements are usually very high level, so any estimate is correspondingly imprecise. As a project moves into later phases, however, more precise techniques can be used to calibrate estimates against the experience and conditions so far for that individual project.

Using the same process throughout is likely to have the same magnitude of error.

Note that the direct output of such a technique, however, would be a p50 estimate—which, with the wealth of historical data available, should be easy to convert into a p10 target if desired (as reported, the ratio was about 2.0, so p10 would be half of the p50), or as a p10 to p90 range.

It would be interesting to get an update from Todd if there's any improvement from using a more accurate, project-calibrated estimation technique later in projects.

Finally, for any software engineers who would

like to improve their estimation skills, I highly recommend Steve McConnell's recent book *Software Estimation: Demystifying the Black Art* (Microsoft Press, 2006). It might be too light for serious estimation experts, but it summarizes the field well for the rest and provides numerous practical tips to help the average company improve its results.

Stephen Gryphon  
Consultant  
Gryphon Technology  
sgryphon@computer.org



I enjoyed very much the piece you published on the Cone of Uncertainty, which I found stimulating, as I still think of software estimation as a "black art." However, I have a few objections to Todd Little's exposé:

1. By substituting duration for effort, he's mixing apples and oranges. While it's true that effort is proportional to duration in a project with constant staff, this isn't the case across a population of projects. We've known since the mid-1970s that these aren't linearly correlated. So, from a statistical perspective, they shouldn't be substituted one for another in the same graph, as in figures 1, 2, and 3. COCOMO and other models have shown that duration equals the square root or cubic root of effort.
2. To truly compare to the Cone of Uncertainty in figure 4, figure 5 should have plotted the inverse—estimated over actual—to make them "similar."
3. Little seems to say that uncertainty isn't reduced, contrary to what we have believed. And he uses the relative uncertainty on the remaining part of the project to make his point. I, for one, had never understood the

We welcome your letters. Send them to software@computer.org. Include your full name, title, affiliation, and email address. Letters are edited for clarity and space.

cone to mean anything other than the absolute overall uncertainty. And using the “time until next milestone” plotted together with the estimated remaining time in figure 7 is again mixing up apples and oranges.

If Little has access to the staffing level in the landmark data, some of these issues would be easy to correct.

*Philippe Kruchten*  
Professor  
University of British Columbia  
[kruchten@ieee.org](mailto:kruchten@ieee.org)

I first saw this article in July 2003 when it was titled “Agility, Uncertainty, and Software Project Estimation.” The simple summary of Landmark Graphics’ estimation results is interesting and is consistent with other data sets my company has seen from our clients—that is, it has many factor-of-two estimation errors. But the data in the article doesn’t support Little’s conclusions about the Cone of Uncertainty.

There are two major problems with the article’s analysis of the Cone of Uncertainty. First, Little doesn’t account for the effect of iterative development on the Cone. In his July 2003 draft of the article, he emphasized that the projects in the data set were using agile practices, and in particular that they emphasized responding to change over performing to plan and experienced significant requirements churn. It’s unfortunate that the version of the article that *IEEE Software* published didn’t capture this feature of the project data set.

If the projects averaged 329 days and they were following agile practices as Little described in the 2003 version, there could easily be five to 10 iterations within each project. But the Cone applies to single iterations of the requirements-design-build-test process. For an analysis of the Cone of Uncertainty to be meaningful in a highly iterative context, the article would need to account for the effect of iteration on the Cone by looking at each iteration separately (that is, by looking at lots of little Cones). If each data point in the article was a summary of multiple iterations, but not detailed iteration-by-iteration data, the ar-

ticle’s comments about the Cone are, unfortunately, meaningless.

The second problem is that the article fundamentally misunderstands the point of the Cone. As I’ve emphasized when I’ve written about it, the Cone represents *best-case* estimation accuracy; it’s easily possible to do worse—as many organizations have demonstrated for decades. Little’s right that the Cone doesn’t guarantee improved estimation accuracy; it’s a hope, but not a promise.

*Steve McConnell*  
CEO and chief software engineer  
Construx Software  
[steve.mcconnell@construx.com](mailto:steve.mcconnell@construx.com)

#### *Todd Little responds:*

I appreciate the interest in my article and the opportunity to clarify several points.

**On the question of how duration data can be compared to effort data.** In all of my analysis, the real issue is the comparison of the ratio of the actual to the estimate for either effort or duration. The ratio of ac-

tual over estimated effort is related to the ratio of actual over estimated duration by the ratio of actual over estimated average staff size. The question, then, is how does staff size change as it becomes apparent that the project was over- or underestimated? In our case, and in what I believe to be a high percentage of situations in software product companies, staff size was fixed by budgetary considerations and not increased. When that’s the case, then the ratio of actual over estimated effort is identical to the ratio of actual over estimated duration.

Kruchten’s comment about the CO-COMO relation of duration and estimation is similar to statements McConnell has written—that because duration has been shown to be proportional to the cube root of effort, duration uncertainty could be reduced to the cube root of effort uncertainty if staff is added to the project. While this statement might be true in an ideal world, I think it misstates the issue.

Let’s assume that a project’s effort was underestimated by a factor of 4.0,

PORTLAND STATE  
UNIVERSITY

# OMSE ONLINE

"We Fit Education into *Your* Life"

Get an online graduate degree or certificate through  
the Oregon Master of Software Engineering program



OMSE ONLINE classes  
start September 25

For more information:

www.omse.pdx.edu • omse@omse.pdx.edu • 1-800-547-8887 ext. 2902

and it proceeded with the originally estimated staff profile. Let's further assume that staff is added incrementally over the life of the project. Let's further assume that this staff addition could be made, and let's also ignore any Brook's Law implications. For the project duration to be underestimated only by a factor of 1.6 (the cube root of 4.0), this would require increasing the originally estimated average staff requirement by more than a factor of 4.0 by the end of the project. I certainly haven't seen anything like that on projects that are run by software product companies.

To reiterate, the estimation process at Landmark essentially guaranteed that effort was proportional to duration. Staff size was based on existing team size, which in turn was based on a product line budget established on the basis of product profitability. It was extremely rare for teams to add or remove staff during these projects. In general, the ratio of actual to estimated average staff was nearly 1.0.

#### **On the question of iterative development.**

While it's true that, as a software product company, we deemed it critical to "respond to change over following a plan," it's not the case that all these projects were run using short-cycle iterative development. Also, even when projects were run with iterations, the project managers felt comfortable classifying the project into the four Microsoft Solution Framework phases: envisioning, planning, developing, and testing.

In any event, I don't see a lot of value in evaluating cones for a particular iteration. For strictly time-boxed iterations, such a cone for duration uncertainty would be meaningless. What's of interest isn't any individual iteration, but rather the ultimate release to the customer base. Figures 5 and 6 in my article properly represent the overall release using dimensionless time.

The primary difference between figure 5 and Boehm and McConnell's traditional Cone is that Boehm uses software phases and figure 5 uses dimensionless time. Figure 5's obvious drawback is that it can't be drawn until the project is over because dimensionless time depends on

the actual delivery date. However, because figures 6 and 7 show that the relative uncertainty band is essentially unchanging, there are still useful guidelines for determining remaining uncertainty.

#### **On the question of the Cone of Uncertainty as a best case.**

I don't have or know of any data to support or refute that the traditional Cone of Uncertainty is a best case. Actually, the cone that I present in figure 5 establishes the worst-case boundary scenario. At a point halfway to the ultimate release date, it's not possible to be off by more than a factor of 2. The upper bounding area is a hyperbola function of  $y = 1/x$ , and the lower bounding area is given by  $y = x$ .

#### **On the question of expert judgment and incorporating project knowledge.**

There's nothing inherent about expert judgment that prevents learning within a project or across projects. In fact, you could argue that expert judgment actually has a better chance of incorporating learning from the project than other estimation methods, assuming the same experts are involved throughout the project. Studies by Magne Jørgensen and others have shown that expert judgment is as good as, if not better than, other approaches. I would welcome any study at a software product company that demonstrates that alternative estimation approaches have resulted in improved estimation or more successful products.

#### **On the question of improving estimates over time.**

It has often been stated that you can't really improve something until you start measuring it. I published an article a year ago in *IEEE Software* ("Context-Adaptive Agility: Managing Complexity and Uncertainty," May/June 2005) on how some changes to our development process and philosophy, aligned with agile software development approaches, let us manage uncertainty in a manner that tightened our differential between estimate and actual, as evidenced by an increase in EQF (estimation quality factor) from 4.8 to 8.4.

**On the unasked question of why estimation is important (or not).** Software effort and

duration estimation is certainly an emotional issue, as you can see by the strong response to this article. My purpose in writing the article was to provide some depth of understanding for practicing software teams. As a practitioner in a software product company, I'm concerned that as an industry, we're looking at the wrong issue. We've let ourselves be talked into a definition of success that's primarily based on being "on time" rather than maximizing value delivery. I contend that "on time" is a poor proxy definition of success.

We can look at Microsoft WinWord 1.0 as an example. By the on-time definition, this project was a terrible failure. Yet I contend that it was one of the most successful projects in the history of software development. Why? The fact that I'm writing this letter using much of the technology developed in that release should give a clue. This project produced a product, and that product has generated substantial business value.

Because business value is a lagging indicator, it's difficult to measure. There's an old story about a man looking for his car keys under a street light. A stranger drops by and asks him where he dropped his keys. He responds by saying he dropped them over by the bench. When the stranger asks why he's not looking over by the bench, the man responds that the light is much better where he's looking now. We seem to do the same thing in software engineering. It's much easier for us to make and compare cost estimates than it is for us to estimate and measure value. But just because it's easier doesn't mean that we should focus on it to the exclusion of the value side of the equation. I encourage others to see if we can improve our understanding of value generation. To this end, I offer an article I wrote two years ago in *IEEE Software* ("Value Creation and Capture: A Model of the Software Development Process," May/June 2004), along with the several other excellent articles in that issue on return on investment. ☺

Todd Little  
Senior development manager  
Landmark Graphics  
tlittle@lgc.com