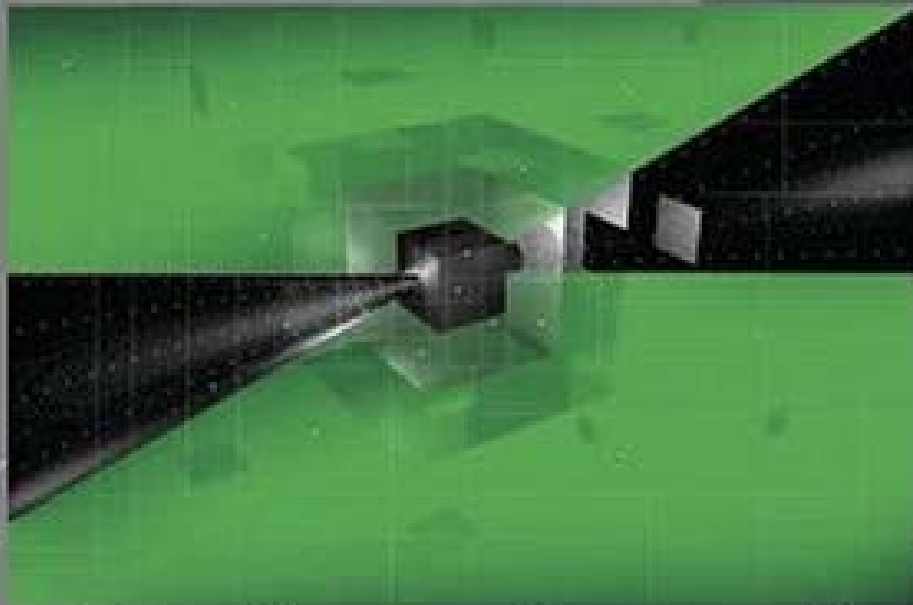


Microsoft

BEST PRACTICES

SOFTWARE ESTIMATION



Demystifying the Black Art

Steve McConnell

Two-time winner of Software Development magazine's Jolt Award

This material is excerpted from *Software Estimation: Demystifying the Black Art* by Steve McConnell (Redmond, Wa.: Microsoft Press).

© 2006 All Rights Reserved.

Chapter 7

Count, Compute, Judge

Applicability of Techniques in This Chapter

	Count	Compute
What's estimated	Size, Features	Size, Effort, Schedule, Features
Size of project	S M L	S M L
Development stage	Early-Late	Early-Middle
Iterative or sequential	Both	Both
Accuracy possible	High	High

Suppose you're at a reception for the world's best software estimators. The room is packed, and you're seated in the middle of the room at a table with three other estimators. All you can see as you scan the room are wall-to-wall estimators. Suddenly, the emcee steps up to the microphone and says, "We need to know exactly how many people are in this room so that we can order dessert. Who can give me the most accurate estimate for the number of people in the room?"

The estimators at your table immediately break out into a vigorous discussion about the best way to estimate the answer. Bill, the estimator to your right, says, "I make a hobby of estimating crowds. Based on my experience, it looks to me like we've got about 335 people in the room."

The estimator sitting across the table from you, Karl, says, "This room has 11 tables across and 7 tables deep. One of my friends is a banquet planner, and she told me that they plan for 5 people per table. It looks to me like most of the tables do actually have about 5 people at them. If we multiple 11 times 7 times 5, we get 385 people. I think we should use that as our estimate."

The estimator to your left, Lucy, says, "I noticed on the way into the room that there was an occupancy limit sign that says this room can hold 485 people. This room is pretty full. I'd say 70 to 80 percent full. If we multiply those percentages by the room limit, we get 340 to 388 people. How about if we use the average of 364 people, or maybe just simplify it to 365?"

Bill says, "We have estimates of 335, 365, and 385. It seems like the right answer must be in there somewhere. I'm comfortable with 365."

84 **Part II Fundamental Estimation Techniques**

“Me too,” Karl says.

Everyone looks at you. You say, “I need to check something. Would you excuse me for a minute?” Lucy, Karl, and Bill give you curious looks and say, “OK.”

You return a few minutes later. “Remember how we had to have our tickets scanned before we entered the room? I noticed on my way into the room that the handheld ticket scanner had a counter. So I went back and talked to the ticket taker at the front door. She said that, according to her scanner, she has scanned 407 tickets. She also said no one has left the room so far. I think we should use 407 as our estimate. What do you say?”

7.1 Count First

What do you think the right answer is? Is it the answer of 335, created by Bill, whose specialty is estimating crowd sizes? Is it the answer of 385, derived by Karl from a few reasonable assumptions? Is it Lucy’s 365, also derived from a few reasonable assumptions? Or is the right number the 407 that was counted by the ticket scanner? *Is there any doubt in your mind that 407 is the most accurate answer?* For the record, the story ended by your table proposing the answer of 407, which turned out to be the correct number, and your table was served dessert first.

One of the secrets of this book is that you should avoid doing what we traditionally think of as estimating! If you can *count* the answer directly, you should do that first. That approach produced the most accurate answer in the story.

If you can’t count the answer directly, you should count something else and then *compute* the answer by using some sort of calibration data. In the story, Karl had the historical data of knowing that the banquet was planned to have 5 people per table. He *counted* the number of tables and then computed the answer from that.

Similarly, Lucy based her estimate on the documented fact of the room’s occupancy limit. She used her *judgment* to estimate the room was 70 to 80 percent full.

The least accurate estimate came from, Bill, the person who used only *judgment* to create the answer.

Tip #30

Count if at all possible. *Compute* when you can’t count. Use *judgment* alone only as a last resort.

7.2 What to Count

Software projects produce numerous things that you can count. Early in the development life cycle, you can count marketing requirements, features, use cases, and stories, among other things.

In the middle of the project, you can count at a finer level of granularity—engineering requirements, Function Points, change requests, Web pages, reports, dialog boxes, screens, and database tables, just to name a few.

Late in the project, you can count at an even finer level of detail—code already written, defects reported, classes, and tasks, as well as all the detailed items you were counting earlier in the project.

You can decide what to count based on a few goals.

Find something to count that's highly correlated with the size of the software you're estimating If your features are fixed and you're estimating cost and schedule, the biggest influence on a project estimate is the size of the software. When you look for something to count, look for something that will be a strong indicator of the software's size. Number of marketing requirements, number of engineering requirements, and Function Points are all examples of countable quantities that are strongly associated with final system size.

In different environments, different quantities are the most accurate indicators of project size. In one environment, the best indicator might be the number of Web pages. In another environment, the best indicator might be the number of marketing requirements, test cases, stories, or configuration settings. The trick is to find something that's a relevant indicator of size in your environment.

Tip #31

Look for something you can count that is a meaningful measure of the scope of work in your environment.

Find something to count that's available sooner rather than later in the development cycle The sooner you can find something meaningful to count, the sooner you'll be able to provide long-range predictability. The count of lines of code for a project is often a great indicator of project effort, but the code won't be available to count until the very end of the project. Function Points are strongly associated with ultimate project size, but they aren't available until you have detailed requirements. If you can find something you can count earlier, you can use that to create an estimate earlier. For example, you might create a rough estimate based on a count of marketing requirements and then tighten up the estimate later based on a Function Point count.

Find something to count that will produce a statistically meaningful average Find something that will produce a count of 20 or more. Statistically, you need a sample of at least 20 items for the average to be meaningful. Twenty is not a magic number, but it's a good guideline for statistical validity.

Understand what you're counting For your count to serve as an accurate basis for estimation, you need to be sure the same assumptions apply to the count that your historical data is based on and to the count that you're using for your estimate. If you're counting marketing requirements, be sure that what you counted as a "marketing requirement" for your historical data is similar to what you count as a "marketing requirement" for your estimate. If your historical data indicates that a past project team in your company delivered 7 user stories per week, be sure your assumptions about team size, programmer experience, development technology, and other factors are similar in the project you're estimating.

Find something you can count with minimal effort All other things being equal, you'd rather count something that requires the least effort. In the story at the beginning of the chapter, the count of people in the room was readily available from the ticket scanner. If you had to go around to each table and count people manually, you might decide it wasn't worth the effort.

One of the insights from the Cocomo II project is that a size estimation measure called Object Points is about as strongly correlated with effort as the Function Points measure is but requires only about half as much effort to count. Thus, Object Points are seen as an effective alternative to Function Points for estimation in the wide part of the Cone of Uncertainty (Boehm et al 2000).

7.3 Use Computation to Convert Counts to Estimates

If you collect historical data related to counts, you can convert the counts to something useful, such as estimated effort. Table 7-1 lists examples of quantities you might count and the data you would need to compute an estimate from the count.

Table 7-1 Examples of Quantities That Can Be Counted for Estimation Purposes

Quantity to Count	Historical Data Needed to Convert the Count to an Estimate
Marketing requirements	<ul style="list-style-type: none"> ■ Average effort hours per requirement for development ■ Average effort hours per requirement for independent testing ■ Average effort hours per requirement for documentation ■ Average effort hours per requirement to create engineering requirements from marketing requirements

Table 7-1 Examples of Quantities That Can Be Counted for Estimation Purposes

Quantity to Count	Historical Data Needed to Convert the Count to an Estimate
Features	<ul style="list-style-type: none"> ■ Average effort hours per feature for development and/or testing
Use cases	<ul style="list-style-type: none"> ■ Average total effort hours per use case ■ Average number of use cases that can be delivered in a particular amount of calendar time
Stories	<ul style="list-style-type: none"> ■ Average total effort hours per story ■ Average number of stories that can be delivered in a particular amount of calendar time
Engineering requirements	<ul style="list-style-type: none"> ■ Average number of engineering requirements that can be formally inspected per hour ■ Average effort hours per requirement for development/test/documentation
Function Points	<ul style="list-style-type: none"> ■ Average development/test/documentation effort per Function Point ■ Average lines of code in the target language per Function Point
Change requests	<ul style="list-style-type: none"> ■ Average development/test/documentation effort per change request (depending on variability of the change requests, the data might be decomposed into average effort per small, medium, and large change request)
Web pages	<ul style="list-style-type: none"> ■ Average effort per Web page for user interface work ■ Average whole-project effort per Web page (less reliable, but can be an interesting data point)
Reports	<ul style="list-style-type: none"> ■ Average effort per report for report work
Dialog boxes	<ul style="list-style-type: none"> ■ Average effort per dialog for user interface work
Database tables	<ul style="list-style-type: none"> ■ Average effort per table for database work ■ Average whole-project effort per table (less reliable, but can be an interesting data point)
Classes	<ul style="list-style-type: none"> ■ Average effort hours per class for development ■ Average effort hours to formally inspect a class ■ Average effort hours per class for testing
Defects found	<ul style="list-style-type: none"> ■ Average effort hours per defect to fix ■ Average effort hours per defect to regression test ■ Average number of defects that can be corrected in a particular amount of calendar time
Configuration settings	<ul style="list-style-type: none"> ■ Average effort per configuration setting

Table 7-1 Examples of Quantities That Can Be Counted for Estimation Purposes

Quantity to Count	Historical Data Needed to Convert the Count to an Estimate
Lines of code already written	<ul style="list-style-type: none"> ■ Average number of defects per line of code ■ Average lines of code that can be formally inspected per hour ■ Average new lines of code from one release to the next
Test cases already written	<ul style="list-style-type: none"> ■ Average amount of release-stage effort per test case

Tip #32

Collect historical data that allows you to compute an estimate from a count.

Example of counting defects late in a project Once you have the kind of data described in the table, you can use that data as a more solid basis for creating estimates than expert judgment. If you know that you have 400 open defects, and you know that the 250 defects you've fixed so far have averaged 2 hours per defect, you know that you have about 400×2 equals 800 hours of work to fix the open defects.

Example of estimation by counting Web pages If your data says that so far your project has taken an average of 40 hours to design, code, and test each Web page with dynamic content, and you have 12 Web pages left, you know that you have something like 12×40 equals 480 hours of work left on the remaining Web pages.

The important point in these examples is that *there is no judgment in these estimates*. You count, and then you compute. This process helps keep the estimates free from bias that would otherwise degrade their accuracy. For counts that you already have available—such as number of defects—such estimates also require very low effort.

Tip #33

Don't discount the power of simple, coarse estimation models such as average effort per defect, average effort per Web page, average effort per story, and average effort per use case.

7.4 Use Judgment Only as a Last Resort

So-called expert judgment is the least accurate means of estimation. Estimates seem to be the most accurate if they can be tied to something concrete. In the story told at the beginning of this chapter, the worst estimate was the one created by the expert who used judgment alone. Tying the estimate to the room occupancy limit was a little better, although it was subject to more error because that approach required a judgment about how full the room was as a percentage of maximum occupancy, which is an opportunity for subjectivity or bias to contaminate the estimate.

Historical data combined with computation is remarkably free from the biases that can undermine more judgment-based estimates. Avoid the temptation to tweak computed estimates to conform to your expert judgment. When I wrote the second edition of *Code Complete* (McConnell 2004a), I had a team that formally inspected the entire first edition—all 900 pages of it. During our first inspection meeting, our inspection rate averaged 3 minutes per page. Realizing that 3 minutes per page implied 45 hours of inspection meetings, I commented after the first meeting that I thought we were just beginning to gel as a team, and, in my judgment, we would speed up in future meetings. I suggested using a working number of 2 or 2.5 minutes per page instead of 3 minutes to plan future meetings. The project manager responded that, because we had only one meeting's worth of data, we should use that meeting's number of 3 minutes per page as a guide for planning the next few meetings. We could adjust our plans later based on different data from later meetings, if we needed to.

Nine hundred pages later, how many minutes per page do you think we averaged for the entire book? If you guessed 3 minutes per page, you're right!

Tip #34

Avoid using expert judgment to tweak an estimate that has been derived through computation. Such "expert judgment" usually degrades the estimate's accuracy.

Additional Resources

Boehm, Barry, et al. *Software Cost Estimation with Cocomo II*. Reading, MA: Addison-Wesley, 2000. Boehm provides a brief description of Object Points.

Lorenz, Mark and Jeff Kidd. *Object-Oriented Software Metrics*. Upper Saddle River, NJ: PTR Prentice Hall, 1994. Lorenz and Kidd present numerous suggestions of quantities that can be counted in object-oriented programs.

