Practical Guidelines for Expert-Judgment-Based Software Effort Estimation

Magne Jørgensen, *Simula Research Laboratory*

Simple process changes such as reframing questions can lead to more realistic estimates of software development efforts.

Improving software effort estimation doesn't necessarily require introducing sophisticated formal estimation models or expensive project experience databases. The following story shows that it can be as simple as reframing questions to more accurately capture the project's context and characteristics.

A software development team estimating the effort needed to complete a complex project divided it into activities, estimated the effort required for each, and eventually arrived at a total of about 17,000 work hours—20,000 maximum, 15,000 minimum. Then the team leader asked a member with considerable domain expertise what his experience had been on similar software projects. "I recall spending at least 40,000 work hours," the developer said. "Even then, about 25 percent of them failed completely. In all projects, more than 50 percent of the effort was due to unexpected events."

From this seasoned developer's answer to a simple, almost off-the-cuff question, this team realized their estimate was hugely overoptimistic, and the risk of total failure was much higher than they'd previously thought. This happens all too often in our field. To try to address this, I've distilled seven guidelines for producing realistic software development effort estimates. The guidelines derive from industrial experience and empirical studies. While many other guidelines exist for software effort estimation,[1] these guidelines differ from them in three ways:

- They base estimates on expert judgments rather than models.
- They are easy to implement.
- They use the most recent findings regarding judgment-based effort estimation.

Estimating effort on the basis of expert judgment is the most common approach today,[2] and the decision to use such processes instead of formal estimation models shouldn't be surprising. No substantial empirical evidence favors using formal estimation models, which are typically more complex and less flexible. Among 15 empirical studies comparing the estimation accuracy of expert judgment and formal models, five favored expert estimation, five found no difference, and five favored formal models.[2]

A Revised Terminology

pX effort estimate: The effort level at which the estimator believes there's X percent probability of *not* exceeding the estimated effort. The goal when providing a pX effort estimate should be accuracy alone. A project leader might, for example, believe that 20,000 work hours is the project's p50 effort estimate—that is, there's a 50 percent probability of not exceeding 20,000 work hours.

Planned effort: The effort used in the project plans. For example, a project leader wishing to emphasize a low risk of overrunning the plan might use a p80 effort estimate. On the other hand, a project leader who believes that lower planned effort leads to more efficient work might use a p30 effort estimate. Typically, the planned effort derives from a combination of goals related to accuracy, project management, and work efficiency.

Effort-to-win: The effort accepted by a client or market—for example, through a bidding round. The main concerns when assessing effort-to-win are related to market price and financial profit. The pX effort estimate is useful input when deriving the profit potential—that is, determining how likely it is that the project will be profitable.

Taken together, the guidelines here don't provide a complete estimation process or discuss all relevant variables, such as the impact of schedule compression or the importance of project management skills. They simply aim to support the expert judgment process with some useful, easily implemented practices. Although fairly general, not all will prove relevant for all software organizations and projects. An organization considering changing its software estimation processes according to these guidelines should therefore consider carefully how well each fits its environment and goals.

Don't mix estimation, planning, and bidding

What is a software development effort estimate? Is it the most likely effort, the effort with a 50-percent probability of not exceeding the estimate, the planned effort, the effort used as a basis for the bid, or something else? People use the term "effort estimate" for all these purposes, even within the same organization and the same project.

But estimation, planning, and bidding have different goals. When estimating the most-likely effort, the goal must be accuracy alone. Planning, however, should also focus on the planned effort's impact on work efficiency and the risks of overrunning the schedule. Bidding goals relate to winning bidding rounds and profit.

Many software organizations have no distinct processes and tools for estimating, planning, and bidding, but merging these processes ultimately reduces the estimate's realism.[2] This reduction might be largely unconscious—for example, client expectations can strongly affect most-likely effort estimates.[3] Yet software professionals I've interviewed believed such expectations hadn't affected them at all, or only slightly. Knowing that the clients' price expectations are unrealistic doesn't eliminate their effects on the estimate. Two ways exist to avoid these effects:

- Don't give the person estimating a project's most-likely effort any pricing-related information, such as the expected price-to-win.
- Always treat planning and bidding as processes separate from effort estimation.

The "A Revised Terminology" sidebar defines terminology that supports independent estimation, planning, and bidding processes and improves the communication of estimation-related information. A recommended technique is the *pX estimate*. It defines the effort level that the estimator believes has an X percent probability of not exceeding the estimated effort, and it starts with the *p50 effort estimate* of most likely effort (see the sidebar "Estimating pX Effort"). An alternative method starts with an estimate of most optimistic effort,[4] but earlier experiments showed this doesn't significantly affect accuracy, and little reason exists for changing the common practice.

Combine estimation methods

One of the most robust findings in forecasting, human judgment, and software estimation studies is that "combination works."[5] Apparently it doesn't matter whether the combination involves a simple average of estimates from different methods or a sophisticated weighting algorithm. A simple average offers a robust combination method unless one estimation method or expert is obviously more reliable than another. As shown elsewhere,[6] though, an expert's technical skill level can be a poor indicator of accuracy, and it's rarely obvious, in advance, which expert will be the better estimator. This is one reason a simple average of outputs from different estimation experts and methods frequently offers the most robust and accurate combination method.

Estimating pX Effort

You can derive the pX effort estimate from the *p50 effort estimate* (estimate of most likely effort) and the *distribution of previous effort estimation error* using the following method.

First, find the p50 effort estimate. Many estimation methods are available for this purpose, including some based on expert judgment[1] and others based on models.[2] For example, the estimate might reflect the typical effort of a set of completed projects similar to that being estimated (analogy based). Or, it might be based on decomposing the project into activities and estimating each activity (bottom-up).

Next, develop the distribution of estimation error in similar projects. Start by identifying a set of projects with uncertainty levels similar to the project you want to estimate. These projects need not closely resemble the one you're estimating.[3] It's more important that the project set be large. In fact, one proper set seems to be all recent projects in an organization. Then develop an estimation error distribution for these projects. Derive the estimation errors by comparing actual efforts with the p50 effort estimates, not with the planned efforts or the efforts used as the client price basis.

Finally, derive the pX effort estimate by calculating it as the p50 effort estimate plus the estimation error, where previous projects' pX have an equally large or lower error. The estimation team shouldn't adjust the pX estimates on the basis of their own expert judgments of high X values. Although current evidence doesn't argue against expert judgment when estimating the p50 effort estimate, it does argue against it when providing, for example, p80 effort estimates.

So, to find the p80 effort estimate, first find the p50 effort estimate: suppose the typical effort spent on similar projects has been about 30,000 work hours, and you decide to use this figure as the p50 effort estimate. Next, develop the estimation error distribution for similar projects. Examining recently completed projects' estimation error, you find that 20 percent had less than 10 percent effort overrun, 50 percent had less than 25 percent overrun, 80 percent had less than 50 percent overrun, 90 percent had less than 100 percent overrun, and all projects had less than 200 percent effort overrun (see Figure A).

You might seek the p80 effort estimate for budgeting purposes—that is, a contingency buffer leading to an 80 percent probability that the project won't go over budget. Applying the distribution of estimation effort error (Figure A), you find that 80 percent of previous projects had effort overruns below 50 percent. So your p80 effort estimate is 30,000 work hours × 150 percent = 45,000 work hours.

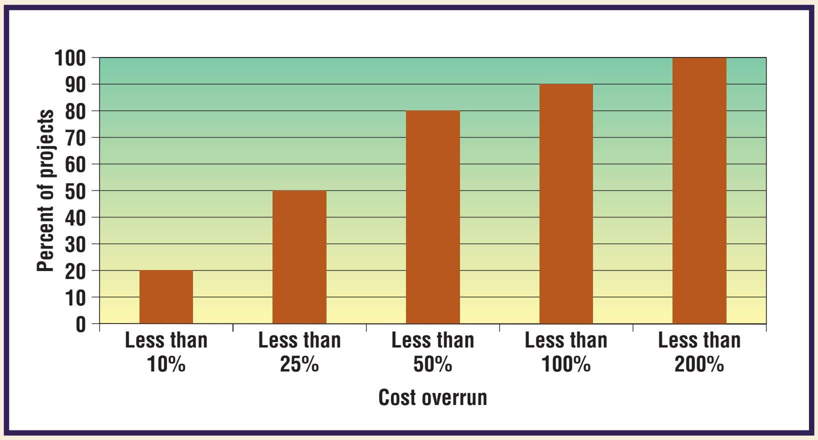

Figure A. Frequency of cost overruns.

References

1. M. Jørgensen, "A Review of Studies on Expert Estimation of Software Development Effort," *J. Systems and Software*, vol. 70, nos. 1–2, 2004, pp. 37–60.
2. B. Boehm et al., *Software Cost Estimation with Cocomo II*, Prentice-Hall, 2000.
3. M. Jørgensen and D.I.K. Sjøberg, "An Effort Prediction Interval Approach Based on the Empirical Distribution of Previous Estimation Accuracy," *J. Information and Software Technology*, vol. 45, no. 3, 2003, pp. 123–136.

When combining estimates, it's most important that the experts and methods represent different estimation principles and knowledge. Fairly obviously, combining estimates with the same strengths or weaknesses also combines the same biases and won't increase the final estimate's accuracy.

However, *confidence* in the estimate's accuracy increases with the number of estimation sources, regardless of the principles upon which they are based. This is why combining estimation methods based on similar principles or judgments of experts with similar backgrounds doesn't increase estimation accuracy but might lead to overconfidence in it. Figure 1 lists examples of useful and ineffective combinations of p50 effort estimates.

An evaluation of this guideline in a Web development company showed that estimation error dropped from 54 percent to 35 percent when estimation teams solicited the opinions of people in different organizational roles.[7]

Ask for justification

A senior project leader who had overseen several large industrial plant development projects that included large software development projects had two golden rules for estimating costs on the basis of expert input:

- Never accept cost estimates based solely on gut feelings.
- Always require justification of cost estimates, so that they can be reviewed.

Several empirical studies support these rules.[8] Knowing that a p50 effort estimate requires justification leads to better estimation processes. One study found that the only factor correlated with software organizations' estimation accuracy was the estimator's accountability.[9] Requiring that estimators justify their results is a good way of increasing their accountability.

I've observed that in many software development projects, effort estimates have no justification other than a reference to one or more software developers' opinions. When the estimation teams decomposed the project into activities and did a requirements analysis, they frequently used a structured and rational process. However, the step from identifying an activity to providing the p50 effort estimate of that activity was frequently based on "magic." The number appeared seemingly from nowhere. A lack of justification means that huge software investments are based on a few experts' judgments that are not reproducible and are impossible to review. Figure 2 lists examples of acceptable and unacceptable effort estimation justifications.

Useful combinations of estimation methods

- Top-down and bottom-up methods
- Analogy and linear regression methods
- Expert judgments and formal methods
- Expert judgments made by software professionals with different project experiences
- Expert judgments made by software professionals in different roles

Ineffective combinations of estimation methods

- Expert judgments from software professionals with similar project experience—for example, team members with experience from the same projects
- Expert judgments from software professionals with the same educational background and organizational role—that is, estimates based on the combination of judgments only from people in technical roles
- Estimation methods based on the same underlying principles, such as two regression-based estimation models with similar variables

Figure 1. Combining p50 effort estimates.

Acceptable justifications

- Reference to actual effort of similar projects
- Breakdown of the project into activities, with reference to effort for performing similar activities in other projects and to the proportion of effort spent on unplanned activities in projects with a similar level of uncertainty
- Application of validated, organization-specific relationships (formulas) between actual effort—preferably with documentation of the validation process

Unacceptable justifications

- "I believe the effort will be 5,250 work hours."
- "The client will not accept a higher cost."
- "We (the estimation team) agreed that the effort would be around project characteristics, such as number of user screens to implement and the 6,500 work hours, and we have a lot of experience."
- Reference to output from a noncalibrated estimation model with no organization-specific data on its performance—use of a noncalibrated, black-box, formal model might actually be "gut feelings" in disguise, where the model input is adjusted so the output fits the "gut feeling"

Figure 2. Justifying p50 effort estimates.

Select estimation experts with experience from similar projects

Experience is more narrowly applicable than most people think,[2] and expertise in "knowing how" differs from expertise in "knowing how much it will cost" and "knowing the uncertainty of the effort estimates." In one company, I found that accurate p50 effort estimates depended mainly on the estimator's ability to recall very similar projects. Overall experience, on the other hand, was a poor indicator of estimation accuracy. In another company, I found that estimates from experts with very similar project experience had on average only 12 percent estimation error. In the other projects, the estimation error averaged 39 percent.[10]

Empirical findings therefore suggest a narrow interpretation of "expertise" and the importance of identifying real experts. Specifically,

- Interpret estimation expertise as experience on very similar projects, not as general software development expertise.

- Consider expertise in knowing how to perform development tasks as different from expertise in knowing how much effort is required to complete them. In fact, technically skilled estimators are sometimes more optimistic and consequently less skilled in effort estimation than non-technically-skilled personnel.[6]
- Select estimators carefully. If no software professionals inside the development organization have experience with similar projects, seek expertise outside the organization. While it's impossible to overestimate the value of relevant estimation experience, it's easy to overestimate the relevance of the estimation experience available.

Accept and assess the uncertainty of effort usage

Software projects frequently have cost overruns. No doubt, we could often avoid or reduce this through better estimation processes and project management. But software organizations and clients must understand that many software projects have high inherent uncertainty and no entirely accurate effort estimate is possible. In such projects we should, rather, learn to accept and assess the high uncertainty instead of denying it or believing we can reduce all of it. Of course, knowing the effort uncertainty level helps us prepare proper contingency buffers and other important means of managing uncertainty. Surprisingly, few guidelines exist for assessing software development effort uncertainty.

A recent evaluation of alternative approaches for assessing uncertainty had several important findings.[11] First, existing formal models for assessing software development effort uncertainty, such as regression-based models, aren't very useful. The formal models, though unbiased, use available information about uncertainty inefficiently. Second, human-judgment-based uncertainty assessments tend to be overconfident, typically assessing effort estimates to be more accurate than they actually are. A recent study found that when a project leader claims to be 90 percent certain (or "almost certain") that the actual effort will not exceed a maximum effort, the actual probability is typically 60 to 70 percent.[3]

To improve uncertainty assessments, we need to move away from the traditional approaches based on the program evaluation and review technique. PERT approaches first estimate the most likely effort (that is, the p50 effort), then the minimum and maximum effort to ensure that, for example, it's 90 percent certain the actual effort will fall between the minimum and maximum effort. This approach can lead to overconfidence, with overly narrow minimum-maximum intervals.

To improve uncertainty assessments, we need to move away from traditional PERT approaches.

An alternative approach:

1. Estimate the p50 effort estimate (most likely effort).
2. Mechanically derive minimum and maximum values—for example, set the maximum effort to 150 percent and the minimum to 90 percent of the estimated most likely effort (someone other than the estimator can set minimum and maximum values based on planning, budgeting, or other information needs).
3. Estimate the probability that the actual effort falls within the minimum-maximum interval.

Two large software companies that applied this method improved their uncertainty assessments' realism from, on average, 16 percent overconfidence to no overconfidence at all.[11]

Provide learning opportunities

Studies on software cost overruns and how experience affects software estimations show little learning from experience alone.[2] This shouldn't surprise us, given the few learning opportunities for effort estimation in software development projects. Typically, a project team estimates a project's most likely effort, then executes the project and measures the actual effort. The actual effort often exceeds the estimated most likely effort. What can we learn from such feedback, other than we have been overoptimistic, again?

Studies on the learning process show that learning depends on understanding causes and effects, professional guidance, and immediate feedback.[12] Performance seems to relate much more to the amount of training than to length of experience. The typical estimation and project execution situation contains few elements required for estimation learning.

Deliberate training (see the sidebar "Deliberate Training for Software Development Estimation") provides more efficient learning because it goes beyond on-the-job estimation training.[13] Among its advantages:

Deliberate Training for Software Development Estimation

I envision a three-phrase training process. The first is the *preparation phase*, in which the organization collects information about a set of completed software projects. A proper set of information includes

- the initial software requirement specification;
- the project experience report describing unexpected events and perceived reasons for accurate/inaccurate effort estimates; and
- the actual effort of the project, separated into activities and following a structure similar to the work breakdown structure typically applied in the organization's estimation phase.

Organize the projects into a meaningful learning sequence, starting with representative but not terribly complex projects, slowly increasing their complexity.

In the *estimation phase*, the trainee receives the requirements specification and other relevant information typically available for estimation. The task is to find the project's p50 effort estimate and uncertainty distribution by applying proper estimation processes. The estimator should document every estimation step and assumption, and experienced project leaders should provide guidance on the first estimation attempts.

In the *feedback phase*, when the trainee has estimated the project and documented the steps and assumptions, an experienced project leader should provide (or supervise provision of) immediate feedback tailored to the trainee's needs. Feedback should include, for example, reasons for inaccurate estimates, such as forgotten activities and incorrect assumptions, and ways to improve the estimation process.

After finishing estimation and feedback on one project, move to the next project in the training sequence. Conduct this learning loop until the trainee's estimation skills are perceived to be sufficient.

- Estimators can assess their experience's validity on different project types, such as much larger projects than they estimated earlier.
- Estimators can analyze reasons for forgotten activities or underestimated risks immediately, rather than establishing biases based on hindsight.
- It's easier to understand the tendency to be overconfident, given proper coaching and training projects.
- Estimators can systematically study the consequences of changes in the estimation process, such as the use of a new estimation method, without risk of estimation failure for ongoing projects.

I know of no scientific evaluations of deliberate learning in software estimation contexts. Nevertheless, convincing findings in other contexts,[13] together with the almost total lack of estimation learning from on-the-job experience observed elsewhere,[14] lead me to believe that deliberate training is worth trying.

Consider postponing or avoiding effort estimation

No textbooks I know of discuss how effort estimates might adversely affect project performance. Some empirical studies do suggest that estimates' mere existence may lower efficiency and that unrealistic estimates can adversely affect project behavior.[14,15] Providing effort estimates based on limited information increases the risk that both the estimates and plans will be overoptimistic.[16] I've observed that attempts to adhere to an overoptimistic estimate lead to insufficient effort spent on analysis, design, and quality assurance. Unplanned analysis and design iterations, in turn, cause project chaos and perhaps large effort overruns during testing and integration. Also, an overoptimistic early effort estimate can easily act as an anchor for subsequent estimates and adversely affect estimators' ability to be realistic when more information becomes available.[3]

In short, good reasons sometimes exist for postponing effort estimation for as long as possible, and even for not estimating at all where the risks of inaccurate estimating might outweigh the benefits of having an estimate. Of course, working without effort estimates doesn't mean planning is unimportant. Good definitions of the project goals, milestones, and deliverables are all significant success factors for software projects, regardless of whether effort estimates exist.

To download further documentation of these guidelines and other useful estimation information, go to the BEST project home page at www.simula.no. The project pages include access to an extensive library of software cost estimation research and estimation checklists.

Acknowledgments

Thanks to Stan Rifkin for his excellent support and comments on this article.

References

1. A.L. Lederer and J. Prasad, "Nine Management Guidelines for Better Cost Estimating," *Comm. ACM*, vol. 35, no. 2, 1992, pp. 51–59.
2. M. Jørgensen, "A Review of Studies on Expert Estimation of Software Development Effort," *J. Systems and Software*, vol. 70, nos. 1–2, 2004, pp. 37–60.
3. M. Jørgensen and D.I.K. Sjøberg, "The Impact of Customer Expectation on Software Development Effort Estimates," *J. Project Management*, vol. 22, no. 4, 2004, pp. 317–327.
4. T. DeMarco and T. Lister, *Waltzing with Bears: Managing Risk on Software Projects*, Dorset House, 2003.
5. R.T. Clemen, "Combining Forecasts: A Review and Annotated Bibliography," *Int'l J. Forecasting*, vol. 5, no. 4, 1989, pp. 559–583.
6. K. Moløkken-Østvold and M. Jørgensen, "Expert Estimation of the Effort of Web-Development Projects: Why Are Software Professionals in Technical Roles More Optimistic Than Those in Non-Technical Roles," *Empirical Software Eng.*, vol. 10, no. 1, 2005, pp. 7–30.
7. K. Moløkken-Østvold and M. Jørgensen, "Group Processes in Software Effort Estimation," *J. Empirical Software Eng.*, vol. 9, no. 4, 2004, pp. 315–334.
8. R. Hagafors and B. Brehmer, "Does Having to Justify One's Judgments Change Nature of the Judgment Process?" *Organizational Behaviour and Human Decision Processes*, vol. 31, no. 2, 1983, pp. 223–232.
9. A.L. Lederer and J. Prasad, "A Causal Model for Software Cost Estimating Error," *IEEE Trans. Software Eng.*, vol. 24, no. 2, 1998, pp. 137–148.
10. M. Jørgensen, "Top-Down and Bottom-Up Expert Estimation of Software Development Effort," *J. Information and Software Technology*, vol. 46, no. 1, 2004, pp. 3–16.
11. M. Jørgensen, "Realism in Assessment of Effort Estimation Uncertainty: It Matters How You Ask," *IEEE Trans. Software Eng.*, vol. 30, no. 4, 2004, pp. 209–217.
12. F. Bolger and G. Wright, "Assessing the Quality of Expert Judgment: Issues and Analysis," *Decision Support Systems*, vol. 11, no. 1, 1994, pp. 1–24.
13. K.A. Ericsson and A.C. Lehmann, "Expert and Exceptional Performance: Evidence of Maximal Adaptation to Task Constraints," *Ann. Rev. of Psychology*, vol. 47, 1996, pp. 273–305.
14. M. Jørgensen and D.I.K. Sjøberg, "Impact of Experience on Maintenance Skills," *J. Software Maintenance and Evolution: Research and Practice*, vol. 14, no. 2, pp. 123–146.
15. D.R. Jeffery and M.J. Lawrence, "Managing Programmer Productivity," *J. Systems and Software*, vol. 5, no. 1, 1985, pp. 49–58.
16. M. Jørgensen and D.I.K. Sjøberg, "Impact of Effort Estimates on Software Project Work," *Information and Software Technology*, vol. 43, no. 15, 2001, pp. 939–948.

About the Author

Magne Jørgensen is a professor of software engineering at the University of Oslo and a member of the software engineering research group of Simula Research Laboratory. His research interests include software cost estimation and expert judgment. He has a Dr.Scient. degree in informatics from the University of Oslo and 10 years of industrial experience. Contact him at Simula Research laboratory, No-1325 Lysaker, Norway; magne.jorgensen@simula.no.