Editor: **Tore Dybå**
SINTEF
tore.dyba@sintef.no

Editor: **Helen Sharp**
The Open University, London
h.c.sharp@open.ac.uk

What We Do and Don't Know about Software Development Effort Estimation

Magne Jørgensen

OVERWHELMING EVIDENCE documents a tendency toward cost and effort overruns in software projects. On average, this overrun seems to be around 30 percent.[1] Furthermore, comparing the estimation accuracy of the 1980s with that reported in more recent surveys suggests that the estimation accuracy hasn't changed much since then. (The only analyses that suggest a strong improvement in estimation accuracy are those by the Standish Group. The extreme improvement in estimation accuracy over the years suggested by its Chaos Reports is, however, probably just a result of an improvement in its own analysis methods from a selection overrepresented by problem projects to a more representative selection.[2]) Estimation methods haven't changed much either. In spite of extensive research on formal estimation models, the dominating estimation method is still expert estimation.[3]

An apparent lack of improvement in estimation accuracy doesn't mean that we don't know more about effort estimation than before. In this article, I try to summarize some of the knowledge I believe we've gained. Some of this knowledge has the potential of improving estimation accuracy, some is about what most likely will not lead to improvements, and some is about what we know we don't know about effort estimation. The full set of empirical evidence I use to document the claims I make in this summary appear elsewhere.[1]

What We Know

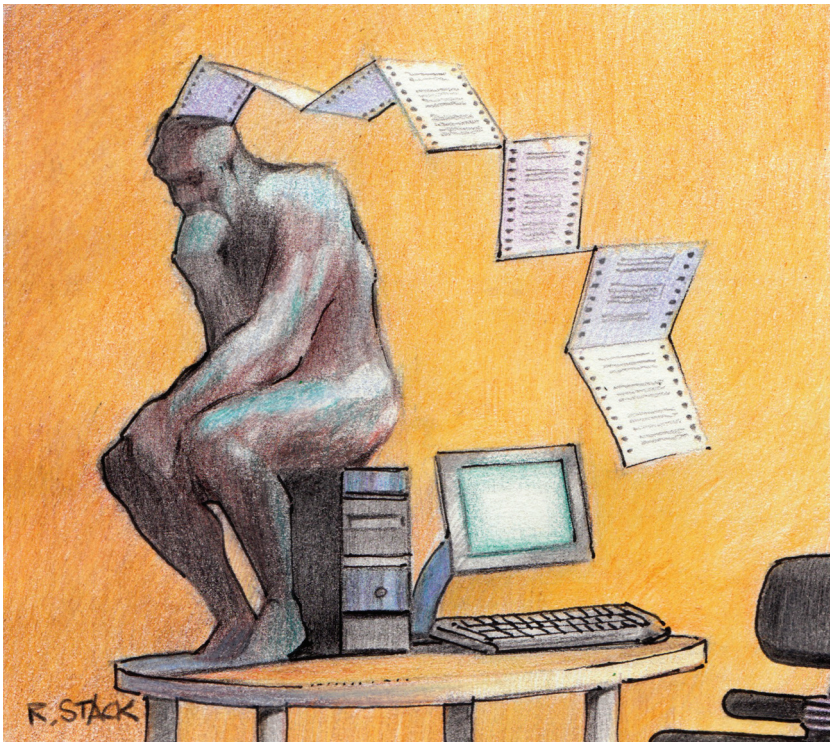
From reviewing the research on effort estimation, I selected seven well-supported results.

There Is No "Best" Effort Estimation Model or Method

Numerous studies compare the accuracy of estimation models and methods, with a large variety in who wins the estimation accuracy contest.[4] A major reason for this lack of result stability seems to be that several core relationships, such as the one between development effort and project size, differs from context to context.[5] In addition, the variables with the largest impact on the development effort seem to vary, indicating that estimation models and methods should be tailored to the contexts in which they're used.

The lack of stability in core relationships might also explain why statistically advanced estimation models typically don't improve the estimation accuracy much, if at all, in comparison

0740-7459/14/$31.00 © 2014 IEEE



with simpler models. Statistically advanced models are more likely to be an overfit to the historical data and will consequently be less accurate than simpler models when the context changes. The findings imply that software companies should try to build their own estimation models, rather than expect that general estimation models and tools will be accurate in their specific context.

Clients' Focus on Low Price Is a Major Reason for Effort Overruns

A tendency toward underestimation of effort is particularly present in price-competitive situations, such as bidding rounds. In less price-competitive contexts, such as in-house software development, there are no such tendencies—in fact, you might even see the opposite. This suggests that a main reason for effort overruns is that clients tend to focus on low price when selecting software providers—that is, the project proposals that underestimate effort are more likely to be started. An implication of these observations is that clients can avoid effort overrun by being less price- and more competence-focused when selecting providers.

Minimum and Maximum Effort Intervals Are Too Narrow

Minimum–maximum effort intervals, such as 90 percent confidence intervals, are systematically too narrow to reflect the actual uncertainty in effort usage. In spite of the strong evidence documenting our inability to set accurate minimum and maximum effort values, current estimation methods continue to assume that it can be done. This is particularly apparent in PERT-based (three-point estimation) methods, in which the planned (median or p50-estimate) effort is derived from the most likely, the minimum, and the maximum effort.

Instead of using expert judgment to set the minimum and maximum effort, software professionals should use historical data about previous estimation error to set realistic minimum–maximum effort intervals.[6]

It's Easy to Mislead Estimation Work and Hard to Recover from Being Misled

All software development effort estimation, even when using formal estimation models, requires expert judgment. But although expert judgment can be very accurate, it's also easily misled. Perhaps the strongest misleading happens when those responsible for the effort estimates, before or during the estimation work, are made aware of the budget, client expectations, time available, or other values that can act as so-called estimation anchors. Without noticing it, those people will tend to produce effort estimates that are too close to the anchors. Knowing that the client expects a low price or a low number of work-hours, for example, is likely to contribute to an underestimation of effort. Expert judgment can also be misled when an estimation request includes loaded words, such as, "How much will this small and simple project cost?"

In spite of much research on how to recover from being misled and how to neutralize estimation biases, no reliable methods have so far been found. The main consequence is that those in charge of effort estimation should try hard not to be exposed to misleading or irrelevant information—for example, by removing misleading and irrelevant information from requirements documentations.

Relevant Historical Data and Checklists Improve Estimation Accuracy

One well-documented way to improve the accuracy of effort estimates is to use historical data and estimation checklists. When the historical data are relevant and the checklist tailored to the company using it, activities are less likely to be forgotten, and it's more likely that sufficient contingency for risk will be added and previous experience reused. This in turn leads to more realistic estimates. In particular, when similar projects can be used in so-called analogy or reference class estimation,[7] effort estimate accuracy improves.

In spite of the usefulness of historical data (such as data about the percentage effort typically spent on unplanned activities and project management) and estimation checklists (such as reminders about easily forgotten activities), many companies currently don't use either tool to improve their estimation accuracy.

Combining Independent Estimates Improves Estimation Accuracy

The average of effort estimates from different sources is likely to be more accurate than most individual effort estimates. A key assumption for accuracy improvement is that the estimates are independent, which is true if their sources differ in expertise, background, and estimation process. A Delphi-like estimation process, such as "Planning Poker," where software developers show their independently derived estimates (their "poker" cards) at the same time, seems to be particularly useful in software effort-estimation contexts.

A group-based, structured estimation process adds value to a mechanical combination of estimates because sharing the knowledge increases the total amount of knowledge, such as the total amount of activities to be completed on a project. The negative effect of group-based judgments, such as "groupthink" and the willingness to take higher risks in groups, isn't documented to be present in software effort estimation.

> Estimates not only forecast the future but also frequently affect it.

Estimation models seem to be, on average, less accurate than expert estimates. The difference between the processes of models and experts make it nevertheless particularly useful to combine these two methods to improve the estimation accuracy.

Estimates Can Be Harmful

Estimates not only forecast the future but also frequently affect it. Too-low estimates can lead to lower quality, possible rework in later phases, and higher risks of project failure; too-high estimates can reduce productivity in accordance with Parkinson's law, which states that work expands to fill the time available for its completion.

This is why it's important to consider whether an effort estimate is really needed. If you don't really need estimates or just need them at a later stage, it might be safer to go without them or postpone the estimation until more information is available. Agile software development—which involves planning just the next sprint or release by using feedback from previous sprints or releases—might be a good way to avoid the potential harm from estimating too early.

What We Don't Know

There are several estimation challenges for which we simply have no good solution, sometimes in spite of volumes of research. Three challenges in particular highlight how our knowledge is far from satisfactory.

How to Accurately Estimate the Effort of Mega-large, Complicated Software Projects

Mega-projects impose extra demand on effort estimation. Not only are more values at stake, but there will also be less relevant experience and historical data available. Many of the activities typical for mega-projects, such as organizational issues with many stakeholders involved, seem to be very hard to estimate accurately because they typically involve business process changes, and complex interactions between stakeholders and with existing software applications.

How to Measure Software Size and Complexity for Accurate Estimation

In spite of the years of research into measuring software size and complexity, none of the proposed measures are

very good when it comes to estimating software development effort. Some size and complexity contexts might enable accurate effort estimates, but such contexts seem to be rare.

How to Measure and Predict Productivity

Even if you have good measures of software size and complexity, you need to reliably predict the productivity of the individuals or teams completing the work. This prediction is complicated by an often surprisingly large difference in productivity among software developers and teams. No good method, except perhaps realistic programming tests (trialsourcing), for this type of prediction exists.

Currently, we don't even know whether there's an economy of scale (productivity increases with increasing project size) or a diseconomy of scale (productivity decreases with increasing project size) in software projects. Most empirical studies suggest that software projects on average have an economy of scale, whereas software practitioners typically believe in a diseconomy of scale. Unfortunately, the research results on scale economies seem to be a consequence of how the analysis is conducted and don't tell much about the underlying relationship between size and productivity.

What we currently know about software effort and cost estimation doesn't really enable us to solve the estimation challenges in the software industry. It does, however, point out several actions likely to improve estimation accuracy. In particular, software companies are likely to improve their estimation accuracy if they do the following:

- Develop and use simple estimation models tailored to local contexts in combination with expert estimation.
- Use historical estimation error to set minimum–maximum effort intervals.
- Avoid exposure to misleading and irrelevant estimation information.
- Use checklists tailored to own organization.
- Use structured, group-based estimation processes where independence of estimates are assured.
- Avoid early estimates based on highly incomplete information.

Highly competitive bidding rounds where the client has a strong focus on low prices are likely to lead to the selection of an overoptimistic bidder and consequently cost overruns and lower software quality. This is termed "the winner's curse" in other domains. In the long run, most software clients might start to understand how their emphasis on fixed, low prices for software projects has negative consequences on project success. Until then, software companies should increase their awareness of when they're in a situation where they're selected only when they're being overoptimistic about the cost—and have strategies in hand for how to manage or avoid the winner's curse.

Avoid early estimates based on highly incomplete information.

References

1. T. Halkjelsvik and M. Jørgensen, "From Origami to Software Development: A Review of Studies on Judgment-Based Predictions of Performance Time," *Psychological Bulletin*, vol. 138, no. 2, 2012, pp. 238–271.
2. M. Jørgensen and K. Moløkken-Østvold, "How Large Are Software Cost Overruns? A Review of the 1994 CHAOS Report," *Information and Software Technology*, vol. 48, no. 4, 2006, pp. 297–301.
3. M. Jørgensen, "A Review of Studies on Expert Estimation of Software Development Effort," *J. Systems and Software*, vol. 70, no. 1, 2004, pp. 37–60.
4. T. Menzies and M. Shepperd, "Special Issue on Repeatable Results in Software Engineering Prediction," *Empirical Software Eng.*, vol. 17, no. 1, 2012, pp. 1–17.
5. J.J. Dolado, "On the Problem of the Software Cost Function," *Information and Software Technology*, vol. 43, no. 1, 2001, pp. 61–72.
6. M. Jørgensen and D.I.K. Sjøberg, "An Effort Prediction Interval Approach Based on the Empirical Distribution of Previous Estimation Accuracy," *Information and Software Technology*, vol. 45, no. 3, 2003, pp. 123–136.
7. B. Flyvbjerg, "Curbing Optimism Bias and Strategic Misrepresentation in Planning: Reference Class Forecasting in Practice," *European Planning Studies*, vol. 16, no. 1, 2008, pp. 3–21.

MAGNE JØRGENSEN works as a researcher at Simula Research Laboratory and a professor at the University of Oslo. His current main research interests include effort estimation, bidding processes, outsourcing, and software development skill assessments. Contact him at magnej@simula.no.