A Critique of How We Measure and Interpret the Accuracy of Software Development Effort Estimation

Magne Jørgensen
Simula Research Laboratory
magnej@simula.no

Abstract

This paper criticizes current practice regarding the measurement and interpretation of the accuracy of software development effort estimation. The shortcomings we discuss are related to: 1) the meaning of 'effort estimate', 2) the meaning of 'estimation accuracy', 3) estimation of moving targets, and 4) assessment of the estimation process, and not only the discrepancy between the estimated and the actual effort, to evaluate estimation skill. It is possible to correct several of the discussed shortcomings by better practice. However, there are also inherent problems related to both laboratory and field analyses of the accuracy of software development effort estimation. It is essential that both software researchers and professionals are aware of these problems and their implications for the analysis of the measurement of effort estimation accuracy.

1. Introduction

As early as 1980, Boehm and Wolverton [1] wrote about the "*need to develop a set of well-defined, agreed-on criteria for the 'goodness' of a software cost model; the need to evaluate existing and future models with respect to these criteria; and the need to emphasize 'constructive' models which relate their cost estimates to actual software phenomenology and project dynamics*". We here claim that there are essential unsolved problems related to the accuracy of effort estimation measurements and that some of these problems are not due to lack of maturity or training, but to inherent problems that may be impossible to solve.

1.1 Problems with MMRE and PRED

The typical current practice when comparing estimation models or evaluating the estimation performance of software organizations in field settings is to apply the accuracy measures Mean Magnitude of Relative Error (MMRE) and PRED or similar measures. MMRE and PRED seem to be have been introduced to the software community by Conte, Dunshmore and Shen in 1985 [2] and are defined as follows:

$$\text{MMRE} = \text{mean MRE} = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{(Est_i - Act_i)}{Act_i}\right|$$

$$\text{PRED(r)} = \frac{k}{n}$$, where k is the number of projects in a set with *n* projects whose MRE <= *r*.

The MRE-based accuracy measures have been criticized by several researchers in software engineering, e.g., [3-5]. Several alternative measures have been proposed, e.g., Mean Balanced Relative Error (MBRE) [6], Weighted Mean of Quartiles of relative errors (WMQ) [7] and Mean Variation from Estimate (MVFE) [8]. Especially illuminating of the problems related to an interpretation of MMRE and PRED as accuracy measures is the paper by Kitchenham *et al*. [4]. That paper proposes that MMRE and PRED should not be interpreted as accuracy measures at all, but instead spread and the kurtosis of the distribution of the estimation accuracy variable z, where z=estimated effort/actual effort.

Conte, Dunshmore and Shen [2] refer to the accuracy measure as mean MRE (MMRE). This is surprising, given that MAPE (Mean Absolute Percentage Error) and MARE (Mean Absolute Relative Error) are the common terms for the same accuracy measure in most other disciplines involving quantitative forecasting. We are not aware of *any* other community that applies the term MMRE for this accuracy measure. This choice of non-standard terminology is unfortunate. It may have contributed to a surprising lack of references in software engineering papers on how to measure and analyze effort estimation accuracy to the large amount of relevant

forecasting, prediction and estimation research papers outside the software engineering community in [9].

Conte, Dunshmore and Shen are also frequently referred to as sources for the claim that effort estimation models should have a MMRE <= 0.2 and a PRED(0.25) >= 0.75. We examined their paper on this topic [2] and found no reference to studies or argumentation providing evidence for these values. Instead we found undocumented claims such as these: "*Most researchers have concluded that an acceptable criterion for an effort prediction model is PRED(0.25) >= 0.75"*. Many researchers have applied these arbitrarily set criterion values to evaluate estimation accuracy of estimation models. We believe that this has reduced the quality of their studies and should be avoided in future studies. The acceptable accuracy of a specific model of processes of judgment depends on many factors, e.g., the accuracy of alternative ways of providing effort estimates in a particular context.

Although there are strong limitations in the accuracy measures themselves, as illustrated by previous papers and our own discussion of MMRE and PRED, it is our opinion that the most severe problems are more basic and, to a large extent, not mentioned in textbooks and papers on software effort estimation. Such problems include those related to what we mean by 'effort estimate' and 'more accurate than', the effect of issues of system dynamics on the meaningfulness of accuracy measurement, and problems related to the outcome-focus of the measurement, i.e., the fact that we are only evaluating the outcome of an estimation process and not the estimation process itself. Neglect of these topics by the software engineering research community and the software industry motivate this paper.

2. What is an 'effort estimate'?

To date, the software development community does not have a precise, agreed upon definition of its most central term, 'effort estimate' [10]. The term 'effort estimate' is sometimes used to mean "planned effort", sometimes the "budgeted effort", sometimes the "most likely use of effort" (modal value), and sometimes the "the effort with a 50% probability of not exceeding" (median value). Sometimes it is not even possible to identify an unambiguous meaning. This confusion may be worst in judgment-based effort estimation (expert estimation), where we have observed that software professionals frequently communicate their effort estimates without stating, or sometimes even being aware of, which interpretation they have used. One consequence of the lack of clear definitions of the term 'effort estimate' is that surveys on the accuracy of software development effort estimation are inherently difficult to interpret; see [11] for an overview. While a 30% effort overrun is likely to result in significant problems of management when the effort estimate was meant to be the planned or budgeted effort, such problems might not arise if it was meant to be the most likely use of effort and the manager added sufficiently large contingency buffers. We seriously doubt the usefulness of surveys where the core concept under investigation, i.e., 'effort estimate', is not precisely defined, inconsistently interpreted, and where there is no information about the degree to which interpretation is inconsistent among the respondents. Unfortunately, this may be the case in most surveys, including our own.

Even in highly controlled situations, e.g., when comparing the estimates of estimation models in laboratory settings, it is frequently not clear what is meant by an effort estimate. For example, it is not clear that the effort estimates derived from analogy-based effort estimation models are of the same type as those derived from regression-based effort estimation models. They have different "loss functions" (optimization functions) and one type of model may, for example, systematically provide higher estimates than others. As a result, an assessment that one estimation model performs better than another may be to the consequence of different interpretations of 'effort estimate'[1]. However, problems related to the interpretation of the term 'effort estimate' are less severe with estimation models than with the more intuition-based expert judgment models because their estimation processes are more explicit.

A reasonable precise interpretation of the term 'effort estimate' is an obvious prerequisite for meaningful measures of estimation accuracy. Without this, it will frequently be difficult to determine whether differences or trends in estimation accuracy result from differences or trends in estimation performance or

[1] Assume, for example, a comparison between analogy and regression-based effort estimation models. The regression-model, due to the least square optimization, will tend to emphasize the historical projects that spent unusually much or little effort compared with the other projects. By contrast, an analogy-based model may assign the same weight to all similar projects when calculating the effort estimate. If the high-impact projects in the data set used for the regression-based estimation models tend to be the projects with unusually high usage of effort, as is frequently the case in software development, the consequence is that the estimates of the regression-based model tend to be higher than those of the analogy-based models. The MRE measures punish over-estimation more than under-estimation. This may mean that the difference in the type of estimates may give the regression model a slight advantage over analogy-based models when applying MRE-based measures.

from different interpretations of 'effort estimate'. As an illustration, estimations of agile projects tend to be based on "how much work can we put into an increment?" rather than the "how much effort will a project require?" of more traditional projects. This means that effort estimates probably play a different role and, possibly, have a different interpretation in agile projects. Thus, comparing estimation accuracy of agile projects with those of traditional projects may be misleading.

3. A moving target

Software development effort estimation is different from many other types of forecasting, e.g., weather or stock price forecasting. While, for example, a weather forecast has no impact on the actual weather and there is typically little doubt what the forecast refers to, the same is not the case in software development projects. In software development projects the target (the required software product) is typically not well-defined, the requirements may change during the project's life-time, and, the estimate itself may impact the process of reaching the target [12]

As an illustration of the implications for the measurement of estimation accuracy, assume that a software provider's estimate of a project is 1000 work-hours and that the price is based on that estimate. Once the project has begun, it soon becomes evident that the estimate of 1000 work-hours was far too low. Hence, the provider either has to face the prospect of huge financial losses or act opportunistically[2] to avoid losses. If the provider acts opportunistically, he might spend less effort on maintainability, usability and robustness properties of the software. As a consequence, the measured estimation accuracy will improve, due to the provider's ability and willingness to adjust the work to fit the initial estimate. Consequently, it is not clear to what extent the measured estimation accuracy measures properties of the estimate or the project's ability to fit the work to the estimate.

The system dynamics of software effort estimation is also important for understanding why it does not necessarily help to 'add 30% to every effort estimate' to remove the bias towards optimism in effort estimation. For example, if a project leader knows that 30% will be added, he may: i) remove 30% from his estimate in advance to get the effort estimate he believes in, or ii) adjust the delivered product to be larger or better. The latter is frequently possible because the effort estimate to some extent defines, as well as reflects, the requirement specifications. Rewarding accurate effort estimates does not work well for similar reasons. For example, we know one company that rewarded those project leaders who had the most accurate effort estimates. The immediate effect was that effort estimates increased and productivity fell. The reason was simply that the project leaders discovered that a simple strategy, rational for them but not necessarily for the company, to achieve accurate effort estimates was to provide higher effort estimates and spend any remaining effort on improving the product ('gold-plating').

In [13] we suggest several methods for measuring estimation accuracy when the target is moving. Unfortunately, it seems that several of the problems involved are inherent and not easy to adjust for. The methods, such as adjusting effort for differences between the specified and actual product, may alleviate problems, but are seldom able to eliminate them completely.

The problems with the measurement of the accuracy of software estimation that result from the target's moving constantly apply mainly in field settings. In laboratory settings, the estimation models are developed from historical data about the completed products and the estimates are derived from those models. In that case, the actual products are used to both develop the model and to evaluate the accuracy of the estimates; hence, there is no moving target problem. However, as soon as we apply and evaluate the models in real-life settings, the problems with respect to measuring estimation accuracy that are induced by the fact that the target may move begin to appear. These problems may strongly restrict the relation of measures of estimation accuracy to properties of the estimates, as opposed to the development process, regardless of the type of accuracy measure chosen.

[2] Opportunistic behaviour (sometimes termed "moral hazard") is a term commonly used in economics. It occurs in software development projects when the provider takes advantage of his superior knowledge about the development processes and software product properties to deliver a product of lower quality than is expected by the client. This would occur, for example, if a provider delivers software products with problems regaridng quality that are not likely to be discovered by the client. The likelihood of opportunistic behaviour increases as the following increase: the degree of information asymmetry between the client and the provider, incentives to deviate from acting in accordance with the clients' goals, and clients' lack of ability to specify and monitor the development process and product. There are controversies regarding to what degree and when opportunistic behaviour will be neutralized by so-called "altruistic behaviour" (the opposite of egoistic behaviour) and "work ethics", but there is no doubt that it is the phenomenon occurs frequently in both software development and other disciplines.

4. What is the meaning of 'more accurate than'?

Preferably, a measure of estimation accuracy should reflect its users' intuitive understanding of important relationships that pertain to accuracy, such as the relationship 'more accurate than'. Otherwise, it may be difficult to find the measure meaningful and to communicate the results. If, for example, most people would agree that the effort estimate X is more accurate than the effort estimate, but our accuracy measure tells us the opposite, there may be problems related to communication and willingness to adopt the accuracy measure.

From discussion with software professionals, we have found that their understanding of 'more accurate than' deviates substantially from that implied by common accuracy measures. The main reason for this is related to the systems dynamics of software development, i.e., the types of reason discussed in Section 3. For example, we have encountered the opinion from a project manager who overestimated a project by 20% that his effort estimate was much more accurate than that of a project that was underestimated by 20%. The reason for the project manager's holding that his estimate was the more accurate is that he knew perfectly well how easily he could have expended exactly as much effort as was estimated, while this option was not available for the project that was underestimated. It is evident that software professionals do not necessarily have a strictly quantitative interpretation of 'more accurate than'.

The lack of a precise and commonly accepted understanding of what we mean by 'more accurate than' may have been a major reason for the acceptance of measures like the MMRE. As an illustration, when we accept that MMRE is a meaningful measure for comparing the estimation accuracy of different estimation methods, we implicitly accept that an effort estimate where the actual effort is 300% of the estimate (MRE = 0.67) is more accurate than an effort estimate where the actual effort is 59% of the estimate (MRE = 0.69). This entails, for example, that if we estimate the effort to be 1000 work-hours and it turns out to be 3000 work-hours, an acceptance of the MMRE measure means that we should agree that we have estimated more accurately than if the actual effort turns out to be 590 work-hours. However, the opposite would probably be the case in practice, we believe.

Other accuracy measures corresponding better to common interpretations of "more accuracy than" and other accuracy relations will hardly ever solve this problem. Even if researchers managed to agree on an 'empirical relational system', i.e., a set of relations and definition sufficiently precise for the purpose of estimation accuracy measurement, it is not reasonable to assume that the software industry will share this 'empirical relational system'. The software industry's loss functions and measurement goals vary a lot, depending on such factors as application domain, person roles, and type of client and project.

We recommend that the estimation accuracy measures be tailored to the situation at hand and, even more importantly, based on a clear understanding of the purpose of the measurement. This may, in several particular situations, lead to precise and meaningful interpretations of relationships that pertain to accuracy, such as 'more accurate than'. Assume, for example, that an organization wants to monitor the proportion of projects that have large estimation overruns to determine whether this proportion increases or decreases over time. The organization must then define precisely what it means by 'large estimation overrun'. They may decide that, for their measurement purposes, situations with 'large estimation overruns' can meaningfully be defined as situations in which the actual effort is more than 30% and more than 500 work-hours higher than the estimated effort, after adjusting the actual effort for differences between planned and actually delivered functionality and quality. By taking these steps, the organization introduces an ordinal scale of estimation overrun measurement, where estimates are categorized and ordered into the categories 'no large effort overrun' and 'large effort overrun'.

If the purpose of the measurement is not only to monitor, but to understand the reasons behind, estimation overruns, several other steps have to be taken, e.g., the steps suggested in [13]. Otherwise, it will not be possible to determine whether a decrease in estimation accuracy within a company is a result of increased estimation skill, change in interpretation of 'effort estimate', less complex projects, or better project management.

5. Focus on the outcome of accuracy measurement

The two main strategies when evaluating the accuracy of judgments are: i) determining coherence with a normative process (the coherence-based strategy) and ii) determining correspondence with the real world (the correspondence-based strategy) [14]. While many studies on human judgment are based on

coherence with a normative strategy, software development effort estimation accuracy evaluations are, as far as we know, based solely on correspondence with the actual effort. A possible reason for this strong reliance on correspondence is that it is frequently not obvious what a normative effort estimation process should look like. However, the disadvantage of this reliance on correspondence is that an effort estimate may be accurate for the wrong reasons.

We are currently analyzing the judgmental processes of 28 software professionals who estimate the effort of software projects and maintenance tasks (work-in-progress). Frequently, we observed, the effort estimation strategies deviated from what we consider to be normative responses, e.g., the response that best reflects the historical data. Sometimes, however, the less defensible estimates were more accurate than those based on normative estimation strategies. Figure 1 displays one such situation.

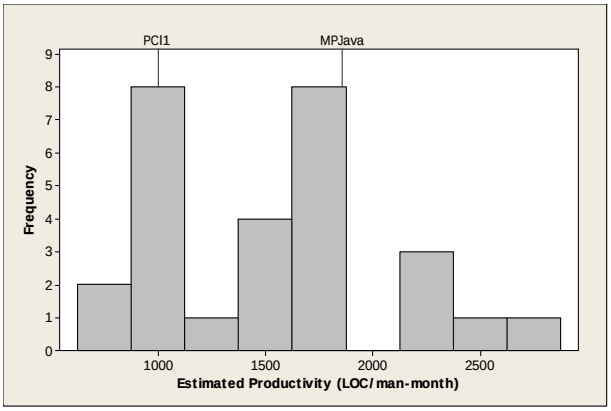

Figure 1: Estimation of programming productivity

In Figure 1, PCl1 denotes the productivity of the closest analogy (the project most similar to the one to be estimated), while MPJava denotes the mean productivity of all previous tasks of same type (five Java tasks). In this estimation situation, the only information about the new task was its estimated size and development platform. This was introduced to force the software professionals to base their estimates on the historical data, so that we could analyze their use of these data.

It is evident from Figure 1 that most software professionals estimated effort values that implied a productivity of the estimated task close to PCl1, MPJava or a combination of these two values. However, there were a few estimates of more than 2000 LOC/man-month. These estimates were difficult to defend on the basis of the historical data received by the software developers. These non-normative estimates may have been caused by: i) calculation error, ii) assumption of organizational learning (not supported by the data), or iii) assumption of much higher productivity of smaller tasks (not supported by the data).

The actual productivity of the task was 2333 LOC/man-month. Hence, those who followed the seemingly least normative estimation strategies had the least discrepancy between estimated and actual effort. This illustrates that if we want to use estimation accuracy as a measure of estimation skill, or predict the future estimation accuracy of a software developer, a strong reliance on outcome without considering the normativeness of the estimation strategy may lead to poor interpretations.

In order to improve the interpretation of estimation accuracy measurement, we recommend that the normativeness of the estimation strategy be evaluated. Elements of an evaluation of the normativeness of an estimation process should include:

- Including variables that historically have affected the use of effort.
- Excluding variables that have had no or very limited impact on the effort.
- Basing the evaluation on a defensible strategy, e.g., similarity to other projects.
- Regressing towards the mean effort or productivity of a larger group of similar tasks with higher uncertainty levels.
- Not assuming substantial learning from experience, i.e., better performance than on previous projects, unless a strong argument for this is provided.

We acknowledge that it is typically very difficult to evaluate the degree of normativeness of judgment-based estimation strategies. There may, for example, be an essential difference between a software professional's claim that he has not been affected by irrelevant variables and the actual effect [15].

6. Final reflections

Common measures of the accuracy of software development effort have, we believe, shortcomings that have a severe impact on their communication, interpretation and meaningful use. Several of these shortcomings have, as far as we are aware, received little attention from the software development communities. We believe that researchers and organizations that apply these effort estimation accuracy measures will benefit from greater awareness of these shortcomings.

We also believe that a necessary precondition for the sustainable improvement of effort estimation is that

the evaluation of judgmental strategies and formal estimation models has a solid foundation. We find that this is currently not the case, and that there is a strong need for more mature analyses and studies on this topic. If we are not able to evaluate and compare estimation models and processes properly, any measure of change may be a result of shortcomings with respect to how we measure, rather than an actual change in estimation performance.

7. References

1. Boehm, B.W. and R.W. Wolverton, *Software cost modeling: Some lessons learned.* Journal of Systems and Software, 1980. **1**: p. 195-201.
2. Conte, S.D., H.E. Dunsmore, and V.Y. Shen, *Software effort estimation and productivity.* Advances in Computers, 1985. **24**: p. 1-60.
3. Shepperd, M., M. Cartwright, and G. Kadoda, *On building prediction systems for software engineers.* Empirical Software Engineering, 2000. **5**(3): p. 175-182.
4. Kitchenham, B.A., et al., *What accuracy statistics really measure.* IEE Proceedings Software, 2001. **148**(3): p. 81-85.
5. Foss, T., et al., *A simulation study of the model evaluation criterion MMRE.* IEEE Transactions on Software Engineering, 2003. **29**(11): p. 985-995.
6. Miyazaki, Y., et al., *Robust regression for developing software estimation models.* Journal of Systems and Software, 1994. **27**(1): p. 3-16.
7. Lo, -.B.-W.-N. and Xiangzhu-Gao, *Assessing software cost estimation models: criteria for accuracy, consistency and regression.* Australian Journal of Information Systems, 1997. **5**(1): p. 30-44.
8. Hughes, R.T., A. Cunliffe, and F. Young-Martos, *Evaluating software development effort model-building techniques for application in a real-time telecommunications environment.* IEE Proceedings Software, 1998. **145**(1): p. 29-33.
9. Jørgensen, M. and M. Shepperd, *A systematic review of software cost estimation studies.* IEEE Transactions on Software Engineering, 2007. **33**(1): p. 33-53.
10. Grimstad, S., M. Jørgensen, and K. Moløkken-Østvold, *Software Effort Estimation Terminology: The Tower of Babel.* Information and Software Technology, 2006. **48**(4): p. 302-310.
11. Moløkken, K. and M. Jørgensen. *A review of software surveys on software effort estimation*, in *International Symposium on Empirical Software Engineering*. 2003. Rome, Italy: Simula Res. Lab. Lysaker Norway.
12. Jørgensen, M. and D.I.K. Sjøberg, *Impact of effort estimates on software project work.* Information and Software Technology, 2001. **43**(15): p. 939-948.
13. Grimstad, S. and M. Jørgensen. *A Framework for the Analysis of Software Cost Estimation Accuracy*. in *ISESE*. 2006. Rio de Janeiro: ACM Press.
14. Hammond, K.R., *Human judgement and social policy: Irreducible uncertainty, inevitable error, unavoidable injustice.* 1996, New York: Oxford University Press.
15. Jørgensen, M., *A review of studies on expert estimation of software development effort.* Journal of Systems and Software, 2004. **70**(1-2): p. 37-60.