Assessing Uncertainty of Software Development Effort Estimates: The Learning From Outcome Feedback

Tanja Gruschke
University of Oslo
tanjag@ifi.uio.no

Magne Jørgensen
Simula Research Laboratory
magne.jorgensen@simula.no

Abstract

To enable properly sized software projects budgets and plans it is important to be able to assess the uncertainty of the estimates of most likely effort required to complete the projects. Previous studies show that software professionals tend to be too optimistic about the uncertainty of their effort estimates. This paper reports the results from a preliminary study on the role of outcome feedback in the learning process on effort estimation uncertainty assessment. Software developers were given repeated and immediate outcome feedback, i.e. feedback about the discrepancy between the estimated most likely effort and the actual effort, for the purpose of investigating how much, and how, they improve (learn). We found that a necessary condition for improvement of uncertainty assessments of effort estimates may be the use of explicitly formulated uncertainty assessment strategies. By contrast, intuition-based uncertainty assessment strategies may lead to no or little learning.

1. Introduction

Surveys of software development project effort estimates report that effort estimates are frequently inaccurate. A recent review of software estimation accuracy surveys [1] suggests that the average effort overrun of software projects is 30-40% and that the accuracy of estimates has not improved over the last 10-20 years. Important reasons for the lack of accurate effort estimates were formulated by Alfred M. Pietrasanta at IBM Systems Research Institute as early as 1968: "*Anyone who expects a quick and easy solution to the multi-faceted problem of resource estimation is going to be disappointed. The reason is clear: computer program system development is a complex process; the process itself is poorly understood by its practitioners; the phases and functions which comprise the process are influenced by dozens of ill-defined variables; most of the activities within the process are still primarily human rather than mechanical, and therefore prone to all the subjective factors which affect human performance*" [2]. As we see it, the problems reported by Pietrasanta are just as valid today as in 1968. Some of the problems seem to be inherent in the estimation of software development effort, which suggests that we should expect a high level of effort estimation inaccuracy in future software development projects, i.e., that there is no reason to believe that the "estimation problem" will be solved any time soon.

A consequence of an expected high level of estimation inaccuracy is that project plans, bids and budgets cannot be based on the cost estimate alone. They must be based on a combination of both the estimates of most likely cost, and knowledge about the likely level of inaccuracies of the estimates. That is to say, knowledge about the uncertainties of the cost estimates is required. Assume that a project leader estimates that the most likely cost of a project is \$ 100 000. The project leader also recognizes that the estimate of most likely cost is quite uncertain and that it is possible that a use of substantially more resources will be required. He therefore decides to base the budget on estimated most likely cost and a contingency buffer. The contingency buffer is included to increase the likelihood that the budget will not be overrun, and the use of it is in accordance with good project management practice [3]. Continuing on our example, the project leader may want to know how likely it is that the project will cost less than \$ 150 000, contingency buffer included. The challenge is then to assess the probability that the project will cost less than \$ 150 000. Not surprisingly, software professionals find such assessments difficult [4]. Unfortunately, as far as we know, there are no alternative proper methods for calculating the size of effort or cost contingency buffers in software projects. In our experience, one cost contingency calculation method frequently applied by the software industry is to set the contingency buffer to a fixed proportion of the project's estimate of the most likely cost, e.g., to set the contingency buffer, as a rule, at 25% of the most likely cost. This practice is not optimal and leads, in highly uncertain projects, to contingency buffers that are too small.

This paper reports from a study on how, and how much, software developers improve their assessments of effort estimation uncertainty (effort prediction intervals) on the basis of typical "on-the-job" feedback, i.e., from a repeated comparison of assessed uncertainty of estimated most likely effort and actual use of effort. For example, we study the learning strategies in situations where the estimator receives feedback that implies that the estimates are systematically more inaccurate than he assessed them to be. The experiments on learning from outcome feedback are conducted in a learning friendly environment, e.g., many small tasks of similar type, well-defined specifications and with immediate feedback. If learning is absent in such environments, this may suggest even less learning in more realistic environments, e.g., large projects with feedback several months after the assessment itself. It must be noted that there may be even more learning friendly environments than the one in our study, as the participants are not trained or instructed how to learn from the feedback. This lack of training and instruction was present, because we were interested in studying how the participants behaved in non-instructed context. The focus on uncertainty assessment learning in favorable conditions also implies that an emphasis of industrial realism has not been emphasized in our experiment. The essential design elements of the described experiment are the learning-friendly environment and that the participants were experts on the type of programming tasks completed. The use of experts were important to avoid that better uncertainty assessment should be attributed to better learning in how to complete the tasks, and not to uncertainty learning. In this experimental design it is not essential that the environment and the subjects are representative for industrial environment and software professionals. The goal is not to generalize to more industrial contexts, but to investigate necessary conditions for uncertainty assessment learning.

The paper is organized as follows: Section 2 briefly describes the terminology related to uncertainty that was used to report the study. Section 3 describes related work motivating our study. Section 4 describes the design of the study. Section 5 reports the results of the study. Section 6 discusses the results. Section 7 concludes.

2. Terminology

The terminology used in contexts of software effort estimation and uncertainty assessment can be confusing. The most important terms are applied as follows in this paper:

Effort estimate: Forecast (predictions) of expected effort. Without any further description, the precise meaning of this term may be unclear, e.g., whether 'estimate' means the 'modal' ('most likely'), the 'median', or, the 'mean' value of a distribution of possible effort usage [5]. We therefore try to avoid this term when we need to be precise, e.g., we use the term '*estimate of most likely effort*' when the modal value of the distribution of possible effort usage is meant.

Effort Uncertainty: A description of the expected uncertainty in use of effort. The type of description of uncertainty applied in this paper is based on *effort prediction intervals*.

Effort prediction interval: A minimum-maximum interval for effort, with a connected confidence level of including the actual effort value. For example, an estimator may estimate the most likely effort to be 1 000 work-hours and the probability of including the actual effort in the effort interval from 600 to 1 500 work-hours to be 90%. Then, the 90% confidence effort prediction interval is [600; 1 500] work-hours. Effort prediction intervals are frequently used in the planning and budgeting of software projects [4].

Estimation outcome feedback: Information about the discrepancy, if any, between the actual effort (the outcome) and the estimated most likely effort. The information about this discrepancy can be used to improve the accuracy of the assessed level of effort prediction intervals. Estimation outcome feedback is frequently the only type of feedback received in software projects, i.e., there is typically no systematic investigation of reasons for higher or lower uncertainty.

3. Related work

In previous studies [4, 6] we found that software professionals strongly underestimated the uncertainty connected with software development effort estimates. For example, when project leaders are 90% sure that the actual development effort would be included by a minimum-maximum effort interval, the typical inclusion rate ("hit rate") is only 60-70%. Other studies of software developers, e.g. [7], report similar results. High levels of over-confidence in estimation accuracy have been documented in other domains [8-11], so underestimation of uncertainty is not limited to software development effort estimation. Several studies [12, 13] report poor estimation learning from the results of previous estimates. This lack of improvement of estimation skills as a result of on-the-job experience seems to be present in most domains, according to Hammond [14, p. 278]: *"Yet in nearly every study of experts carried out within the judgment and decision-making approach, experience has been shown to be unrelated to the empirical accuracy of expert judgments"*.

Two frequently reported reasons for the fact that estimators are poor at learning from their estimation experience are lack of relevant feedback and lack of immediate feedback [15, 16]. Both Gagné [26] and Ericsson et al. [28] identify the activity of feedback, i.e. communication to the learner about the degree of correctness of the performance, as crucial for a controlled learning progress. We, therefore, believe that learning more realistic uncertainty assessment of effort estimation may improve as better feedback is provided. The level of program development skills seems to be a poor indicator of ability to assess realistically the uncertainty of estimates of most likely effort. In [17], for example, we found that: *"The level of over-confidence was higher in situations where at least one of the team members assessed his/her knowledge to be very high…."* Similar results are reported in [18]. In other words, higher development skill may in some situations result in greater over-confidence. For a comprehensive review of studies on software estimation uncertainty assessments see [19]. As a result of the lack of a correlation between amount of experience in making effort estimations and the ability to assess realistically the uncertainty of effort estimates, i.e. poor learning from experience, there is a need for better understanding of the necessary conditions for learning from experience in the context of software effort estimations. The study reported in this paper is a step towards that goal.

4. Design of study

As indicated in Section 3, reasons for the poor learning previously observed in the real-world-setting are not thoroughly understood. For the purpose of finding a method to improve the realism in uncertainty assessment of effort estimation, we believe it is essential to understand how this particular skill evolves, i.e. the learning process of uncertainty assessment. An in-depth study of few subjects solving many tasks, as in this study, is therefore likely to be a better study design than a study design where many subjects solve fewer tasks [25, 27]. The chosen type of study design is frequently used in psychology, psychiatry and education research.

We decided to investigate the relations between uncertainty assessment strategy, feedback, and learning, i.e., the conditions for learning, in a software development task-solving context with rather favorable learning conditions. The relevance of the feedback was ensured by high similarity of the tasks to be solved. The timeliness of the feedback was ensured by providing the feedback immediately after the completion of a task and just before the uncertainty of the effort estimation of a new task was provided. The learning bias towards "hindsight bias" was reduced by the short duration (less than 5 hours) of the task. The experiment was designed specifically to enable learning from feedback in favorable learning situations, i.e. we studied necessary, but not sufficient, conditions for learning.

4.1. Research questions

The two main research questions of this study are:

RQ1: How much do programmers improve their assessments of the uncertainty of estimates of most likely effort on the basis of outcome-related feedback?

RQ2: What is the relation between the learning strategies for improving uncertainty assessments used by the programmers and their ability to learn from feedback?

4.2. Measures

There are no standard measures of uncertainty assessment performance. In an earlier paper [6] we argued that we should differentiate between people's ability to assess the *average level of uncertainty* of a set of tasks and the *relative difference* in uncertainty between different tasks. For the purpose of the study reported in this paper we apply the following definitions and measures:

T = A set of *n* development tasks

$\boldsymbol{ActEff_j}$ = Actual effort required to complete Task *j*

$\boldsymbol{EstML_j}$ = Estimated most likely effort of Task *j*

$\boldsymbol{RE_j}$ = Magnitude of estimation error of task *j*
= $(ActEff_j - EstML_j)/ ActEff_j$

$\boldsymbol{MRE_j}$ = Magnitude of relative estimation error of task *j*
= $|ActEff_j - EstML_j| / ActEff_j$

There are many different measures of estimation error, with different strengths and weaknesses. We have chosen to use the MRE measure as it suits our analysis needs here, as well as being the most widespread measure of estimation error.

$\boldsymbol{Int1_j}$ = [90% of $EstML_j$; 110% of $EstML_j$]

$\boldsymbol{Int2_j}$ = [60% of $EstML_j$; 150% of $EstML_j$]

$\boldsymbol{Int3_j}$ = [50% of $EstML_j$; 200% of $EstML_j$]

The widths, i.e., the percentages, were chosen to reflect a narrow effort interval (Int1), a medium-wide effort interval (Int2), and a wide effort interval (Int3). We applied more than one interval to enable analyses of possible differences in learning effects related to width of interval. These are also the same intervals used in [6], see this study for further elaboration of the interval widths.

$\boldsymbol{Conf1(Int1_j)}$ = The developer's assessed probability (confidence) of including $ActEff_j$ in $Int1_j$

$\boldsymbol{Conf2(Int2_j)}$ = The developer's assessed probability (confidence) of including $ActEff_j$ in $Int2_j$

$\boldsymbol{Conf3(Int3_j)}$ = The developer's assessed probability (confidence) of including $ActEff_j$ in $Int3_j$

AvConfLev1(T) = Average value of $Conf1(Int1_j)$ for tasks j=1..n

AvConfLev2(T) = Average value of $Conf2(Int2_j)$ for tasks j=1..n

AvConfLev3(T) = Average value of $Conf3(Int3_j)$ for tasks j=1..n

HitRateInt1(T) = Proportion of $Int1_j$–intervals that includes $ActEff_j$ for tasks j=1..n

HitRateInt2(T) = Proportion of $Int2_j$–intervals that includes $ActEff_j$ for tasks j=1..n

HitRateInt3(T) = Proportion of $Int3_j$–intervals that includes $ActEff_j$ for tasks j=1..n

Applying these definitions we define the ability to assess the average level of uncertainty as:

Overconfidence(Int1,T) = AvConfLev1(T) – HitRateInt1(T)

Overconfidence(Int2,T) = AvConfLev2(T) – HitRateInt2(T)

Overconfidence(Int3,T) = AvConfLev3(T) – HitRateInt3(T)

We have termed the measure 'Overconfidence', because a positive value indicates overconfidence in the accuracy of the estimate of most likely effort. Consider the following example: Assume that an estimator estimates and assesses the estimation uncertainty of a set of tasks (T). On average, the estimator believes that there is a 50% chance of including the actual effort in Int1 for the set of tasks 1..n. The estimator's average confidence level (AvConfLev1(T)) is then 50%. The proportion of actual effort values included in Int1 is, on the other hand, only 30%, i.e., the HitRateInt1(T) is 30%. Then the level of overconfidence is calculated as the difference between average confidence and inclusion rate of Int1 of the set of tasks in T, i.e., Overconfidence(Int1,T) = 50% - 30% = 20%.

Our measures of ability to assess relative difference of uncertainty between different tasks are defined as follows:

RelUncAbility(Int1,T) = Correlation between $Conf1(Int1_j)$ and MRE_j, for j=1..n

RelUncAbility(Int2,T) = Correlation between $Conf2(Int2_j)$ and MRE_j, for j=1..n

RelUncAbility(Int3,T) = Correlation between $Conf3(Int3_j)$ and MRE_j, for j=1..n

It is, a priori, reasonable to assume that there should be a negative correlation between confidence in the accuracy of the estimate of most likely effort (Conf) and the estimation error (MRE). When the confidence in the accuracy of the estimate of most likely effort is low, we would expect a high MRE, i.e., we expect these measures to give high negative values if the estimator is skilled at assessing the relative difference in effort estimation uncertainty between different tasks.

4.3. Subjects, tasks, and, material

At the University of Oslo we advertised for highly skilled Java programmers and selected the five programmers whom we believed to be the best Java-programmers, based on examination of their CVs and interviews. All participants had programmed several thousands lines of code in Java and had industrial programming experience. For the goal of our study, i.e., to study necessary conditions for uncertainty learning processes in learning-friendly situations, these programmers were considered to be suitable tasks experts. We believe that if these experts could not improve uncertainty assessment in the designed learning-friendly situation, this suggests poor opportunities for learning in more noisy and realistic situations. This belief should, however, be subject to empirical examination. The motivation of the participants, both regarding their participation in the study and their focus on improving their uncertainty assessments, seemed to be high. They were paid standard wages according to education level. They all expressed an interest regarding the theme of the study when applying for the job. Also, students are in the "profession of learning". Therefore, students may be more focused on learning from feedback than software development professionals who may be more constrained by real world project pressures and may have no time or energy to learn anything new.

There were five programmers participating. The experiment size measured in number of tasks completed, i.e., more than 80 tasks, is however relatively large compared to most other software engineering experiments. The low number of participants was based on what we believed was the optimal trade-off between number of participants and tasks solved per participant for the purpose of our study of learning processes.

The programming tasks given in the experiment were small, but typical for programming tasks solved by student programmers, i.e., the participants can be considered experts on this type of tasks. There were 18 tasks in all. Software developers tend to work on tasks within a larger context, i.e., individual tasks are typical dependent on each other and a part of the same software system. In our study we wanted to ensure that any increase in estimation uncertainty assessment accuracy was not caused by better understanding of the software system rather than the uncertainty assessment learning. The experimental tasks in our study are independent of each other and any learning is consequently more likely to be a result of uncertainty assessment learning. The tasks were taken from lower grade courses at the department of Informatics at the University of Oslo, beginners' textbooks on Java, and Java Sun's home pages. Most tasks required GUI or other types of visual solutions, others were text-based. The tasks were similar in type and complexity. The sequence of tasks was similar for all participants (see Section 4.4 for a more detailed description) and there was no obvious increase or decrease in complexity of the tasks. The interdependence was verified when investigating the RE after the study was finished. There was found no improved accuracy of most likely effort. This indicates that there was no programming skill learning from task to task, i.e., this confirms the independency between tasks.

An example of a task is the following:

Task description for "Juke box": *Write a program to select an audio file using the file dialog box, and use three buttons, "Play", "Loop" and "Stop", to control the audio. If you click the Play button, the audio file is played once. If you click the Loop button, the audio file keeps playing repeatedly. If you click the Stop button, the playing stops.*

The experiment was performed in one of the Department of Informatics' computer labs. The lab has work stations with UNIX and a Java programming environment. All the participants had their own student IT user account and were familiar with the computer lab and programming environment. The programming was completed using a text editor (most of them used "Emacs") that the participants had customized to their own needs. The specifications of all the other programming tasks are available by request to the authors.

4.4. The experiment process

The experiment took place over a period of about two weeks. Each participant participated for five work-days. On the first day of the experiment all participants were informed about the experiment and it was emphasized that an important purpose of the experiment was to study how well they were able to improve their effort uncertainty assessments. The participants had received the experiment instructions a couple of days before. The programming tasks, i.e., the requirement specifications, were handed out one at a time. When receiving a programming task the participant read the text, briefly analyzed the problem, and estimated the most likely effort (EstML). The estimated most likely effort was given as input to a web page, which then calculated the effort intervals Int1 ([90%;110%] of EstML), Int2 ([60%;150%] of EstML), and Int3 ([50%;200%] of EstML) in work-hours. We then asked the developers to assess the probability of including the actual effort (ActEff) in each of the effort intervals Int1, Int2 and Int3. Then, the participants solved the task and registered the actual effort spent.

When working on a task the participants could take as many breaks as they liked, but the breaks were not included in the actual effort. When they believed that they had finished a task, the person in charge of the experiment (one of the authors) would decide whether the program qualified as an adequate solution by testing it. If it was not adequate or had bugs, the participants were asked to change or correct the program. Once the task had been completed satisfactorily, the participants commented on their estimation and uncertainty assessment performance based on a comparison of the actual effort with their estimated most likely effort and uncertainty assessments. When the task and the comments were completed, a new task was handed out by the person responsible for the experiment. For practical reasons, i.e., to ensure that a task would be completed within a working day, the sequence of the tasks was slightly different among the participants. This makes the comparison of estimation performance between the subjects more difficult, but we do not believe that it affected to any great extent the learning processes of the participants. The software developers solved between 14 and 18 tasks during the 5 work days they were required to participate.

When all tasks were completed we interviewed the participants about their learning strategies and other issues relevant for understanding the results.

5. Results

5.1. Development and estimation skill

As described in Section 4, the number and sequence of tasks completed by the developers varied slightly for practical reasons. Hence, to compare the development and estimation performance of the developers, we selected a subset of tasks that was completed by all developers (12 tasks) in almost the same sequence. Table 1 shows the total effort (TotEff) in work-hours needed to complete the tasks in work-hours, the median estimation error in % of actual effort (Median MRE), and the average time in minutes spent estimating a task (AvEstTime).

Table 1. Comparison of skill on the 12 tasks completed by all developers

Developer	TotEff	Median MRE	AvEstTime
A	16:48	34 %	6,8
B	19:36	18 %	5,3
C	27:08	35 %	4,8
D	22:49	37 %	5,4
E	24:49	22 %	n.a.

Table 1 indicates that the developers were not very different in development skill., i.e., the total effort (TotEff) spent on the 12 tasks are not very different.This suggests that any differences in uncertainty assessmentsare probably not caused by large differences in development skill. There were some differences regarding estimation skill measured as median MRE, e.g., developers B and E had better estimation accuracy. Interestingly, see Section 5.2, developers B and E were the two developers who showed the least improvement in their uncertainty assessment performance. There are no large differences in average time spent on deriving the estimates of most likely effort and the uncertainty assessments of that estimate. From the interviews we found that all developers applied expert judgment, sometimes supported by decomposition of the task into sub-tasks, to estimate the most likely use of effort. This is the estimation method most commonly applied in the software industry as well [20].

5.2. Ability to assess relative difference in uncertainty

Table 2 shows the correlation between confidence level and estimation accuracy for all developers and all tasks completed by the developers, applying the RecUncAbility measure. Notice that there should be a strong negative correlation if the developers are good at distinguishing between high and low effort uncertainty tasks. A positive correlation means that it is more typical that a high uncertainty task is considered to be a low uncertainty task, and vice versa

Table 2. Correlation between confidence and MRE (all tasks)

Developer	Int1	Int2	Int3
A	0.19	0.24	0.20
B	0.11	-0.07	-0.31
C	-0.33	-0.21	0.18
D	0.16	0.11	n.a.*
E	0.48	0.42	0.21

* All confidence levels were 100%

As can be seen from Table 2, most of the correlations are low and/or positive, which suggest a poor ability to assess the relative difference in uncertainty between the tasks. The only developers who were able, to some extent, to separate high uncertainty from low uncertainty tasks may be developers B and C. A possible reason for this poor ability to separate high and low uncertainty estimates may be that our tasks were relatively similar. In an industrial study of 70 real-life software projects [6], i.e., a situation with more heterogeneous tasks, we found (for Int3) a correlation of -0.26 between confidence level and estimation error. To see if there was any learning from experience we calculated the correlation for the ten last tasks only (see Table 3). We expect that if there had been substantial learning, there would have been more negative correlations and higher negative values. The correlations are low and unsystematic and the number of observations is low. We have, for this reason, not included the p-values and recommend a careful interpretation of the numbers.

The data in Table 3 suggest that some of the developers, i.e., developers A, B, and D, did improve their performance, but not by very much. The improvements may also be due to random variation. Overall, we found that the developers were poorly skilled at separating high and low uncertainty task and that the skill improved only slightly, at best.

Interestingly, the correlation between the time spent on the estimation and the uncertainty assessment of a task and

Table 3. Correlation between confidence and MRE (last 10 tasks)

Developer	Int1	Int2	Int3
A	0.04	-0.10	0.11
B	-0.25	-0.02	-0.45
C	0.11	-0.11	0.25
D	-0.17	-0.05	n.a.*
E	0.47	0.49	0.19

* All confidence levels where 100%

the estimation error (MRE) was better, or just as good as, the correlations in Table 1 and 2. This means that the variance in time spent on the estimation work provided just as good an indicator of the variance of the uncertainty of use of effort as did the developers' uncertainty assessments themselves. This further supports the poor ability of the developers to assess the relative difference in effort uncertainty between tasks in the previous analysis.

5.3. The ability to assess the average level of uncertainty

For the purpose of analyzing the ability to assess uncertainty and the improvement from outcome feedback, we created a "learning graph" for all five developers. The learning graph shows how well the developers were able to use the outcome feedback to adjust their assessment of the average level of effort uncertainty to the real uncertainty. The learning graph was derived as follows:

1) Compare assessed uncertainty (confidence) and actual uncertainty (hit rate) for the first five tasks (Comparison Point 1). The degree of overconfidence at Comparison Point 1 is calculated as Overconfidence(Int1,T), Overconfidence(Int2,T), and, Overconfidence(Int3,T), for T={Task1…Task5}.

2) The comparison at Comparison Point 2 is conducted similarly, but now for T = {Task2…Task6}.

3) Etc.

The rationale for including only the five last tasks in the set of tasks (T) is that it turned out to be a useful number of tasks to study the learning effect. Including fewer tasks would lead to more random variation in the learning curve due to difference in task complexity and including more tasks may have hidden some of the learning progress information. Figure 1-5 depicts the "learning graphs" of the developers A, B, C, D and E. The figures provide a good picture of the correlation between a participant's effort prediction intervals, Int1, Int2 and Int3.

Initially, developer A was strongly overconfident, especially with respect to Int1 (the narrow effort interval), but then improved for both Int1 and Int3 (the wide interval) and soon reached a good accordance between assessed and actual effort uncertainty, i.e., close to 0% overconfidence. The uncertainty assessments related to Int2 (the medium

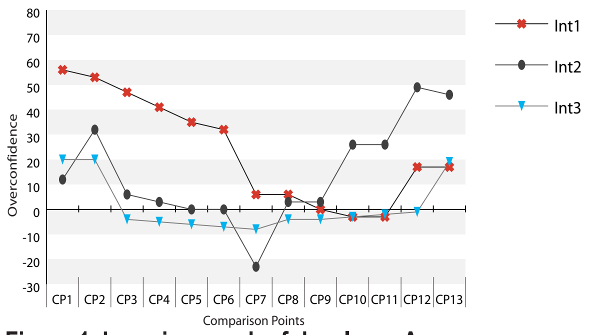

Figure 1. Learning graph of developer A

wide interval), however, went from overconfidence, to underconfidence, and then back to a higher overconfidence than in the beginning.

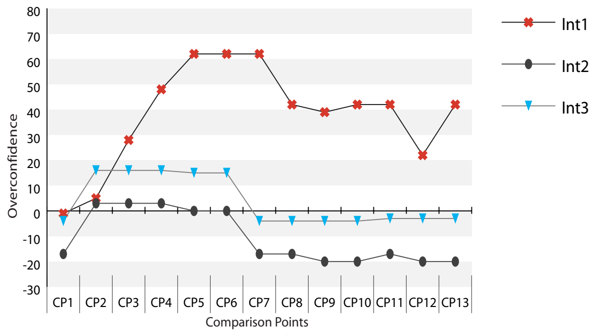

Figure 2. Learning graph of developer B

Developer B had realistic assessments of the probability of including the actual effort in Int2 and Int3 for all comparison points. The developer went, however, from realistic assessment to overconfidence regarding the Int1 effort intervals and there was not much improvement. Notice that developer B had the most accurate effort estimates and had a much higher proportion of actual effort values inside the Int2 interval compared with most of the other developers.

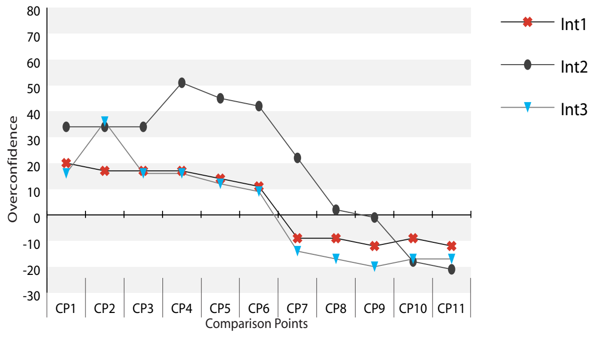

Figure 3. Learning graph of developer C

Developer C went from overconfidence to underconfidence for both Int1, Int2, and Int3. The level of underconfidence is, however, not large and there are clear signs of learning for all intervals.

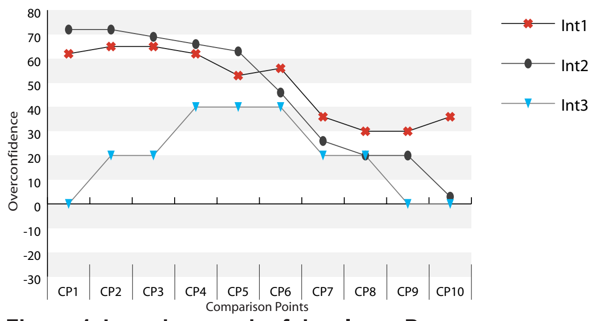

Figure 4. Learning graph of developer D

Developer D started with a high degree of overconfidence of Int1 and Int2, then improved slowly (except for Int3). Developer D had a good correspondence between assessed and actual uncertainty for Int2 and Int3 at the end. Even after completion of all tasks (15 tasks) there was a strong level of overconfidence regarding Int1. However, there are clear signs of (slow) learning.

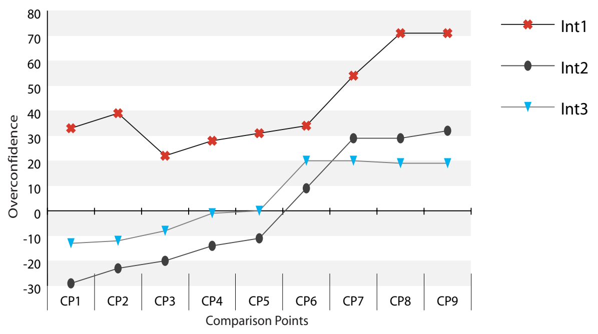

Figure 5. Learning graph of developer E

Developer E went from underconfidence related to Int2 and Int3 to overconfidence. The level of overconfidence related to Int1 increase and at the end is as high as 70%. There are no clear signs of learning.

The developers' ability to learn to assess the uncertainty of the development tasks seems to vary a lot. The performance of developers A, C, and, D suggest a learning from experience, while there was little sign of learning for developers B and E.

5.4. The uncertainty assessment strategies

The strategies for uncertainty assessments and learning described in this section are based on (i) the comments provided by the developers after each task completion, (ii) the interviews with the developers when all tasks were completed, and (iii) a comparison of the comments and interview information with the actual performance. The amount of verbal information and analysis is high and the length restriction on this paper allows only a brief summary of selected topics. The analyses were conducted independently by the two authors of this paper. Only a few minor disagreements needed to be resolved through subsequent discussion and further analysis, i.e., there was a high degree of similarity of the results of the independent analyses of the information related to the use of uncertainty assessment strategies.

Based on earlier experience, we categorized software development estimation effort uncertainty assessments into three main strategies:

S1 (Hit rate-based strategy): Uncertainty assessments through comparison of previous hit rates with current confidence levels. For example, if only 30% of previous actual effort was inside the Int1 effort interval, then the confidence of the next Int1 effort interval should not deviate too much from 30%.

S2 (Analogy-based strategy): Uncertainty assessment through recall of a small set of similar tasks (typically 1-2 tasks) and use of the estimation error of those tasks to set the confidence levels. For example, if the estimation error of the two most similar tasks was about 30%, it is likely that this level of estimation error will occur on the current task as well.

S3 (Intuition-based strategy): Uncertainty assessment without any explicitly formulated strategy, i.e., an intuition-based uncertainty assessment where mainly the properties of the current task is evaluated.

We found that the strategies indicated by the developers in the task comments corresponded well with the confidence levels chosen. The developers' descriptions of their uncertainty assessment strategies in the interviews, on the other hand, did not always correspond well with the two other information sources. All sources are valid, but both authors, independently, formed the impression that the interview-based descriptions were sometimes strongly affected by how the developers wanted the authors to believe that they had conducted the uncertainty assessments. Table 3 summarizes our categorization of the strategies applied by the different developers, according to information source.

The developers' own description of their use of uncertainty assessment strategy in the interviews matched the strategies we derived from the task comments and the analysis of the chosen confidence levels for developers A and D. For the other developers there were deviations. In particular, developer E described his strategy very differently from that which we observed, e.g., there was no indication of the use of S1 in the learning graph in Figure 5.

It is, nevertheless, possible that developer E tried to apply S1, but was unable to apply the feedback properly or used the feedback in a biased manner. However, if that were the case, there should be some evidence of the use of S1 in the task comments. In short, we think that there are

Table 3. Main uncertainty assessment strategies

Devel-oper	Source: Task comments + analyses of chosen confidence levels	Source: Interviews
A	S1	S1
B	S3	S2 + S3
C	S2	S1 + S2
D	S2	S2
E	S3	S1 + S2

reasons to believe that our strategy categorization based on the task comments and analysis of chosen confidence levels is more realistic than the one based on the interviews. The interviews show, nevertheless, that the task of extracting uncertainty assessment strategies is a difficult task and that we may need sophisticated means to increase the validity of the analysis, e.g., the use of think-aloud-protocols.

We also asked the developers about whether they applied mainly analytical or intuitive strategies when making decisions *in general*. Developers A, D, and E perceived themselves as dominantly analytical, while developers B and C perceived themselves as dominantly intuitive decision-makers. Developers A and D stated that they switched to more analytical strategies, because they had experienced that their intuition was frequently very biased. These comments may be interpreted as providing evidence of an ability to be analytic on a meta-level, i.e., to be analytic about choice of decision strategy, which may be a good indicator of learning ability.

Developers A and D were two of the developers that displayed learning in our experiment. The third developer that displayed learning was developer C. Developer C perceived himself as an intuition-based decision-maker, but his uncertainty assessment comments were clearly more analytical than those of developers B and E. Developer C stated in the interviews that he was aware of his tendency to be overoptimistic in his intuition-based decisions and tried to compensate for this. Although this does not show that developer C applies analytical strategies, it shows at least a mature level of reflection about his own biases. This suggests another type of meta-learning ability, i.e., the ability to identify and compensate for one's own biases.

During the interviews there was uncovered that skill level, both in programming abilities and in estimated most likely effort uncertainty assessment, contributes to the build of confidence internally in the developer. We found a subjective interpretation of success rate different from the statistical interpretation, i.e., if the actual effort was close to (but outside) one of the boundaries it was nevertheless considered a hit. For example, actual effort only 5 minutes outside the maximum effort was usually interpreted to be a hit. Very large estimation overruns was frequently considered to be outliers, i.e., not relevant as lessons learned for future assessments. All the developers initially expressed satisfaction over their uncertainty assessments.

When presented a summary of their sometimes poor performance, all developers except developer A sustained this opinion. All of them admitted that they had not kept any statistic of their performance while receiving feedback after each task. None of them looked for historical data (easily available). Some wrote down time spend and/or estimated effort and/or uncertainty levels, but failed to use this historical data actively. In effect, they all relied on their memory when recalling past performance.

This reliance of own memory may have hindered the learning. Einhorn [11] found that unaided use of own memory easily can lead to a "persistence of the illusion of validity" and consequently lead to a too positive assessment of own performance.

6. Discussion of results

As discussed in Section 3, assessment of uncertainty of software development effort estimates has been found to be a difficult task. In our study we designed the task-solving environment and the feedback in a way that we believed would support learning, i.e., the solving of many similar tasks and immediate feedback. Nevertheless, the developers struggled with learning uncertainty assessments from experience and two out of five developers may not have learned at all.

Examining the strategies of the two developers with the poorest learning, we found that their uncertainty assessment strategies were more intuition-based and that they reflected less on their strategies. This suggests that the ability to learn depends on the existence of an explicit uncertainty assessment strategy that improves as more data becomes available. Among the developers there may have been two explicit strategies of that type:

- S1: Adjustment of confidence levels based on hit rate of previous (reasonably similar) tasks. As the number of completed tasks increases, the accuracy of the uncertainty assessments improves. This learning strategy is similar to the learning in the formal uncertainty assessment model we propose in [21].
- S2: Recall of the most similar tasks and use of the uncertainty (estimation error) of those tasks to determine the confidence of the current task. As the set of similar tasks increases, the accuracy of the uncertainty assessments improves.

The lack of improvement in uncertainty assessment when applying intuition-based strategies is in accordance with results from other domains, e.g., the finding that intuition-based assessments may be even more over-confident when a greater amount of information is available [9, 22], and that the bias towards over-confidence is difficult to avoid when processes are intuition-based [23].

Notice that our results do not imply that there actually will be learning from experience in typical software development situations when applying the uncertainty assessment strategies S1 or S2. In [24], for example, we found that many professionals estimation teams had a strong tendency to believe that their effort estimates were accurate, even in situations where they knew that the estimation error in similar occasions typically had been very inaccurate. Our results may, however, demonstrate that *necessary* conditions for learning uncertainty assessments are (i) the use of explicit strategies that improves as more data becomes available, and (ii) non-reliance on intuition-based strategies.

When examining the *threats to validity* of our study one should remember that some of the experimental artificiality was introduced intentionally, to examine learning processes in learning friendly environments, i.e., environments with many similar tasks and immediate feedback. In other words, if we find poor learning in a learning friendly environment, we should expect even poorer learning in a more realistic environment. This is, we believe, a meaningful way of studying necessary conditions for learning, although not for studying sufficient conditions.

We used student programmers in our study. The students were, however, experts on the tasks to be solved and, we believe, useful for the purpose of examination of necessary conditions for learning.

The main problem with our study is, in our opinion, the small number of subjects that we analysed. It is possible that the inclusion of more subjects would reveal that there are, indeed, software developers that are able to improve their uncertainty assessments by applying intuition-based strategies. Our results should mainly be interpreted as hypothesis (or theory)-building results, not as strong evidence in support of the suggested relationships.

7. Summary and further research

Our study aimed at answers to the following two questions: 1) How much do programmers improve their assessments of the uncertainty of estimates of most likely effort from outcome-related feedback?, and 2) What is the relation between the strategies employed by the programmers to learn how to improve uncertainty assessments and their ability to learn from feedback?

To answer these questions, we conducted a study of effort estimation uncertainty ability and learning of five developers who solved between 14 and 18 small (less than five work-hours) programming tasks, with feedback after each task.

We conclude that some of the software developers did have the ability to improve their uncertainty assessments. However, not all developers seem to be able to use the outcome feedback to improve their performance. We hypothesize that a necessary (but not sufficient) condition for learning uncertainty assessment from experience (outcome feedback) is the use of an explicit uncertainty assessment strategy that improves with more feedback.

An implication of the hypothesis is that we cannot expect uncertainty assessments to improve when they are dominantly intuition-based. Our hypothesis is in accordance with the poor effort uncertainty assessment performance and improvement in real software projects [19], and studies from other domains, that show that overconfidence in estimates is difficult to avoid when applying intuition-based strategies [23].

Our study indicates that software companies should instruct software development effort estimators to use explicit uncertainty assessment strategies, such as the strategies S1 and S2 described in Section 5; improved recommendations can be made when the study has been performed on professional software developers.

We have started a replication of this study with software professionals and larger tasks to assess the robustness of the hypothesis posed in this paper.

References

[1] Moløkken, K. and M. Jorgensen. *A review of software surveys on software effort estimation*. in *International Symposium on Empirical Software Engineering*. 2003. Rome, Italy: Simula Res. Lab. Lysaker Norway: pp. 223-230.

[2] Pietrasanta, A.M. *Current methodological research*. in *ACM National Conference (USA)*. 1968: ACM Press, New York, NY, USA: pp. 341-346.

[3] Shenhar, A.J., *One size does not fit all projects: Exploring classical contingency domains.* Management Science, 2001. **47**(3): pp. 394-414.

[4] Jørgensen, M., K.H. Teigen, and K. Moløkken, *Better sure than safe? Over-confidence in judgement based software development effort prediction intervals.* Journal of Systems and Software, 2002. **70**(1-2): pp. 79-93.

[5] Jørgensen, M., *How much does a vacation cost? or What is a software cost estimate?* Software Engineering Notes, 2003. **28**(6): p. 5-5.

[6. Jørgensen, M., *Realism in assessment of effort estimation uncertainty: It matters how you ask.* IEEE Transactions on Software Engineering, 2004. **30**(4): pp. 209-217.

[7] Connolly, T. and D. Dean, *Decomposed versus holistic estimates of effort required for software writing tasks.* Management Science, 1997. **43**(7): pp. 1029-1045.

[8] Klayman, J., et al., *Overconfidence: It depends on how, what and whom you ask.* Organizational Behaviour and Human Decision Processes, 1999. **79**(3): pp. 216-247.

[9] Oskamp, S., *Overconfidence in case-study judgments.* Journal of consulting psychology, 1965. **29**(3): pp. 261 - 265.

[10] Vallone, R.P., et al., *Overconfident prediction of future actions and outcomes by self and others.* Journal of Personality and Social Psychology, 1990. **58**(4): pp. 582 - 592.

[11] Einhorn, H.J. and R.M. Hogarth, *Confidence in judgment: Persistence of the illusion of validity.* Psychological Review, 1978. **85**(5): pp. 395-416.

[12] Lichtenstein, S. and B. Fischhoff, *Do those who know more also know more about how much they know?* Organizational Behaviour and Human Decision Processes., 1977. **20**(2): pp. 159-183.

[13] Jørgensen, M. and D.I.K. Sjøberg, *Impact of experience on maintenance skills.* Journal of Software Maintenance and Evolution: Research and practice, 2002. **14**(2): pp. 123-146.

[14] Hammond, K.R., *Human judgement and social policy: Irreducible uncertainty, inevitable error, unavoidable injustice.* 1996, New York: Oxford University Press.

[15] Shepperd, J.A., J.K. Fernandez, and J.A. Quellette, *Abandoning unrealistic optimism: Performance estimates and the temporal proximity of self-relevant feedback.* Journal of Personality and Social Psychology, 1996. **70**(4): pp. 844-855.

[16] Bolger, F. and G. Wright, *Assessing the quality of expert judgment: Issues and analysis.* Decision support systems, 1994. **11**(1): pp. 1-24.

[17] Jørgensen, M., *Top-down and bottom-up expert estimation of software development effort.* Information and Software Technology, 2004. **46**(1): p. 3-16.

[18] Moløkken, K., *Expert estimation of Web-development effort: Individual biases and group processes (Master Thesis)*, in *Department of Informatics*. 2002, University of Oslo.

[19] Jørgensen, M., *Evidence-Based Guidelines for Assessment of Software Development Cost Uncertainty.* Submitted for publications (an early version can be downloaded from www.simula.no/publication.php

[20] Jørgensen, M., *A review of studies on expert estimation of software development effort.* Journal of Systems and Software, 2004. **70**(1-2): pp. 37-60.

[21]Jørgensen, M. and D.I.K. Sjøberg, *An effort prediction interval approach based on the empirical distribution of previous estimation accuracy.* Information and Software Technology, 2003. **45**(3): pp. 123-136.

[22] Whitecotton, S.M., D.E. Sanders, and K.B. Norris, *Improving predictive accuracy with a combination of human intuition and mechanical decision aids.* Organizational Behaviour and Human Decision Processes, 1998. **76**(3): pp. 325-348.

[23] Wilson, T.D. and N. Brekke, *Mental contamination and mental correction: unwanted influences on judgments and evaluation.* Psychological bulletin, 1994. **116**(1): pp. 117-142.

[24] Jørgensen, M. and K. Moløkken. *Eliminating Over-Confidence in Software Development Effort Estimates,*. in *Conference on Product Focused Software Process Improvement (PROFES)*. 2004. Japan: LNCS 3009, Springer Verlag: pp. 174-184.

[25] Zendler, A., E. Horn, et al. *Demonstrating the usage of single-case designs in experimental software engineering.* Information and Software Technology, 2001. **43**(12): 681-691.

[26] Gagné, R. M. and Medsker, K. L. *The conditions of learning : training applications*. 1996, Fort Worth, Harcourt Brace.

[27] Harrison, W. *N = 1: An Alternative for Software Engineering Research?*, in *22nd International Conference on Software Engineering (ICSE)*. 2000, Limerick, Ireland.

[28] Ericsson, K. A., R. T. Krampe, et al. *The Role of Deliberate Practice in the Acquisition of Expert Performance*, Psychological Review, 1993. **100**(3): 363-406.