Assessing Uncertainty of Software Development Effort Estimates: Learning from Outcome Feedback

Tanja Gruschke[1], Simula Research Laboratory and University of Oslo
Magne Jørgensen[2], Simula Research Laboratory and University of Oslo

Abstract

***Context**: Previous studies report that software developers are over-confident in the accuracy of their effort estimates. **Aim**: This study investigates the role of outcome feedback, i.e., feedback about the discrepancy between the estimated and the actual effort, in improving the uncertainty assessments. **Method**: We conducted two in-depth empirical studies on uncertainty assessment learning. Study 1 included five student developers and Study 2 10 software professionals. In each study the developers repeatedly assessed the uncertainty of their effort estimates of a programming task, solved the task, and received estimation accuracy outcome feedback. **Results**: We found that most, but not all, developers were initially over-confident in the accuracy of their effort estimates and remained over-confident in spite of repeated and timely outcome feedback. One important, but not sufficient, condition for improvement based on outcome feedback seems to be the use of explicitly formulated, instead of purely intuition-based, uncertainty assessment strategies.*

Keywords: D.2 Software engineering, D2.9 Software engineering management, cost estimation, effort prediction intervals, overconfidence

1. Introduction

Surveys of effort estimates for software development projects report that such estimates are frequently inaccurate. A review of surveys on the accuracy of software estimation [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 and that there is no reason to believe that the "estimation problem" will be solved any time soon.

As an illustration of inherent software development effort estimation problems, it may be useful to compare software cost estimation with cost estimation when constructing buildings, i.e., a discipline where the average estimation accuracy is much better. When estimating the cost of construction of buildings the size measures, number of square meters and building height, are known early and strongly correlated to the cost. In [3], for example, a simple regression model based on these two size variables, applied on data from 30 previous

[1] tanjag@simula.no
[2] Corresponding author: magnej@simula.no, Simula Research Laboratory, P.O. Box 134, NO-1325 LYSAKER, NORWAY.

constructions, produced cost estimates of construction of buildings in Hong Kong with a *maximum* error of less than 7%. In comparison, Kitchenham et al. [4] report an *average* software cost estimation error of similarly produced regression models, applying "function points" as size measure, based on a similar number of observations (within the same organization) to be 44%.

A consequence of an expected high level of estimation inaccuracy in software development projects 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 of the likely level of inaccuracy of the estimates. That being so, it is necessary to know the uncertainty of the cost estimates. To see why this is so, consider the following example. Suppose 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 substantially more resources will be required. He therefore considers to base the budget on estimated most likely cost and a contingency buffer of $ 25 000. A contingency buffer is included to increase the likelihood that the budget will not be overrun, and its use is in accordance with good project management practice [5]. Continuing with our example, the project leader now wants to assess how likely it is that the project will cost more than $ 125 000, i.e., how likely it is that the contingency buffer is sufficiently high. Not surprisingly, software professionals find such assessments difficult and are frequently highly over-confident about the accuracy of their cost estimates [6]. An over-optimistic contingency buffer leads easily to poorly managed projects.

This paper reports on two studies of how, and how much, software developers improve their assessments of effort estimation uncertainty on the basis of typical "on-the-job" feedback, i.e., from a repeated comparison of a) estimated effort together with assessed uncertainty (in the format of an effort prediction interval) of the estimate and b) actual error of effort estimates. For example, we studied the learning strategies in situations where the estimator received feedback that implied that his effort estimates were systematically more inaccurate than he assessed them to be. Our studies on learning from outcome feedback were conducted in learning-friendly environments, e.g., there were many small tasks of similar type, well-defined specifications and immediate outcome feedback. If learning is absent in such environments, even less learning is to be expected in more realistic environments, e.g., large projects with outcome feedback that is difficult to compare with the original estimates and uncertainty assessment feedback, received several months after the assessment itself.

The remainder of this paper is organized as follows: Section 2 briefly describes important terminology related to uncertainty assessment. Section 3 describes related work that motivated our studies. Section 4 describes and discusses the designs of the studies. Section 5 reports the result of Study 1. Section 6 reports the results of Study 2. Section 7 discusses and explains the results. Section 8 concludes.

2. Terminology

The most important terms in this paper are used as follows:

Effort estimate: Predictions of the most likely use of effort. The effort estimates in this paper are based on judgmental processes and not the use of formal estimation models.

Effort estimation uncertainty assessment: A description of the expected accuracy of the estimated effort. The type of uncertainty assessment applied in this paper is based on *effort prediction intervals*.

Effort prediction interval: A minimum-maximum interval for effort, with a connected confidence level that the actual value for the effort will be included. For example, an estimator may consider the most likely effort to be 1000 work-hours and the probability of including the actual effort in the effort interval from 600 to 1500 work-hours to be 90%. Then, the 90% confidence effort prediction interval is [600; 1500] work-hours. Effort prediction intervals are used frequently in the planning and budgeting of software projects [6].

Estimation outcome feedback: Information about the discrepancy, if any, between the actual effort (the outcome) and the estimated most likely effort. Information about this discrepancy can be used to improve the accuracy of the confidence that the actual effort will be included in an effort prediction interval. Estimation

outcome feedback in this paper includes information about the discrepancy only, and, no information about possible reasons for the deviation.

3. Related work

In previous studies [6, 7] we found that software professionals underestimated the uncertainty connected with software development effort estimates. For example, when project leaders were 90% sure that the actual development effort would be included in a minimum-maximum effort interval, the typical inclusion rate ("hit rate") was only 60-70%. Other studies of software developers, e.g. [8], report similar results. Underestimation of uncertainty is not limited to software development effort estimation; high levels of overconfidence in estimation accuracy have been documented in other domains [9-11].

Several studies, e.g., [12, 13], report poor learning from estimation experience. This lack of improvement of estimation skills as a result of on-the-job experience seems to be present in many 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 reasons frequently cited for the fact that estimators are poor at learning from their estimation experience are lack of relevant feedback [15], and lack of immediate feedback [16, 17]. Both Gagné [18] and Ericsson et al. [19] identify the activity of feedback, i.e., communication to the learner about the degree of correctness of the performance, as crucial for controlled progress in learning. Hogarth [20] categorizes learning environments as "kind" or "wicked", according to whether or not the feedback they receive is accurate and leads to improved future judgments. Currently, many learning environments in software development organizations may be "wicked", e.g., lack immediate and valid feedback.

One possible solution to the problem of overconfident estimators who do not learn from experience is to replace the estimators with formal effort prediction interval methods. This would remove the bias towards overconfidence, but is not without complications. In a previous study, we concluded on the basis of empirical evidence [21] that ".. *formal approaches and human judgment-based effort prediction intervals seem to have complementary advantages and disadvantages. The experiment indicates that software professionals use the available uncertainty information more efficiently, i.e., have lower median PIWidth* [prediction interval width] *for the same hit rate. On the other hand, they provide effort prediction intervals with a poor correspondence between confidence level and hit rate* [i.e., they are overconfident]. *Efficient use of uncertainty information is particularly important when the number of observations to learn from is low. Then, the application of formal model-based prediction interval approaches may result in meaninglessly wide intervals* … ". Consequently, it is not likely that we will be able to replace judgmental processes with formal uncertainty assessment models any time soon, except in situation with large data sets of relevant observations.

Another possible solution is to select only realistic software developers when we require uncertainty assessments. Unfortunately, we do currently not know how to select software developers with realistic uncertainty assessments. The perceived level of program development skills, for example, may be a poor indicator of the ability to assess realistically the uncertainty of estimates of most likely effort. In [22] we found that: *"The level of overconfidence 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 [23]. In other words, higher development skill may, in some situations, result in greater overconfidence in estimation accuracy and is not necessarily a good predictor of realism. In a recent study [24], we found that the previous level of optimism was the best predictor of future optimism. However, even that indicator would fail to select the more optimistic of two estimators in one third of the cases. There are differences between level of overconfidence and level of optimism, but [24] nevertheless provides additional support for the claim that it is difficult to identify realistic estimators. For a comprehensive review of studies on software development effort estimation uncertainty assessments, see our review in [25]. That review includes, among other issues, a review of results, processes and models for

uncertainty assessment in many disciplines and how they may impact the realism. The uncertainty assessment process we apply in this paper is based on principles support by evidence reviewed in [25].

Our learning goals are the similar to those motivating the Personal Software Process (PSP) [26]. While there are evidence suggesting positive effects this training process with respect to effort estimation accuracy, see for example [27], we have been unable to find any results regarding the use of PSP or similar learning processes for improvement of judgment-based effort estimation uncertainty assessment.

4. Design of studies

In this paper, we focus on better understanding the conditions for learning from experience in the context of uncertainty assessments in software effort estimation. This focus is motivated by the high level of overconfidence, the poor learning from experience, and the lack of obvious alternative solutions to improved learning and training processes, as reported in Section 3.

There are many possible reasons for, and factors leading to, overconfidence and lack of uncertainty assessment learning. To improve the understanding of these reasons and factors we decided to design in-depth studies of a few subjects solving several small programming tasks. This decision was based on the assumption that studying a small number of subjects performing several small tasks would yield more useful results on learning processes than studying more subjects performing fewer but larger tasks; see discussions on this topic in [28, 29].

We investigated the conditions for learning by analyzing the relations between uncertainty assessment strategy, feedback, and observed learning in the context of performing software development tasks under conditions favorable for learning. The relevance of the feedback was ensured by a high degree of similarity in the tasks to be solved. The timeliness of the feedback was ensured by its being provided immediately after the completion of a task. All the tasks performed by a developer were completed within a relative short period of time; typically about five days. The learning bias towards "hindsight bias" was reduced by the short duration of the task, which ensured that there was a short period of time between an uncertainty assessment and the outcome feedback.

Two studies were conducted. The first was conducted with university students who performed programming tasks that were similar to those of which they had previous experience. The second replicated most design elements of the first study, but used software professionals (hired consultants) in a software development setting that approximated a typical real life setting. The most important design differences of the two studies are twofold: 1) the tasks performed were dependent on each other in Study 2 (hence, there may be a learning effect in terms of increased system understanding from one task to another) while being independent of each other in Study 1, and 2) fewer tasks were solved in Study 2 (hence, there was less outcome feedback to learn from.) The research questions and evaluation measures were the same for the two studies. To ensure a sufficiently high motivation for efficient work and high quality solutions we paid the participants normal fees for their work and included a strict quality assurance process. This ensured that all actual effort values related to completion of programs with acceptable quality.

4.1. Research questions

The two main research questions were these:

RQ1: How much do software developers 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 uncertainty assessment strategies used by the software developers and their ability to improve their uncertainty assessments?

Uncertainty assessment improvement (RQ1 and RQ2) is in this paper interpreted as increased correspondence between a developer's confidence in the accuracy of his effort estimates and the actual estimation accuracy. One factor potentially leading to uncertainty assessment improvement is the learning from effort outcome feedback. Measures that enable analysis of uncertainty assessment improvement are described in Section 4.2. The choice of uncertainty assessment strategy (RQ2) is of interest to the learning focus of this paper, because of the potentially close relationship between choice uncertainty assessment strategy and learning (improvement ability) from outcome feedback.

4.2. Measures

For the purpose of the studies reported in this paper we apply the following definitions and measures:

T = A set of tasks estimated and completed by a developer
ActEff = Actual effort required to complete a task
EstML = Estimated most likely effort of a task
RE = Relative estimation error of a task = (ActEff - EstML)/ ActEff
MRE = Magnitude of relative estimation error of a task = |ActEff - EstML| / ActEff

There are many different measures of estimation error, with different strengths and weaknesses. We chose to use the RE and MRE, because they suit our needs and are the most widespread measures of estimation error. This choice of the measures of estimation accuracy is not critical for our analysis and we found that using proposed alternative measures, e.g., measures that replace the denominator ActEff with EstML or min(EstML, ActEff), would not lead to different conclusions.

We defined the following three effort prediction interval types as basis for the uncertainty assessments:

Int1 = [90% of EstML; 110% of EstML]
Int2 = [60% of EstML; 150% of EstML]
Int3 = [50% of EstML; 200% of EstML]

The interval widths, represented by 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 the analysis of possible differences in learning effects related to width of interval, which are the same as those used in [7] (see this study for further elaboration of the selection of interval widths).

A developer's ability to assess the average level of uncertainty of a set of tasks (T) for a given prediction interval type (Int1, Int2 or Int3) is defined as:

Overconfidence(Int,T) = AvConfLev(Int,T) – HitRate(Int,T); where *AvConfLev(Int,T)* is defined as a developer's average confidence (mean assessed probability) of including the actual effort in *Int* for of a set of tasks *T*, and *HitRate(Int,T)* is defined as the frequency of including the actual effort in *Int* for the same set of tasks *T*. Int is either *Int1*, *Int2* or *Int3*.

We have termed the measure of uncertainty assessment ability 'Overconfidence', because a positive value indicates overconfidence in the accuracy of the estimate of most likely effort. Consider the following example. A developer is requested to assess the uncertainty of effort estimate of a set of tasks (T) for the interval Int1, i.e., he is requested to assess the probability that the actual effort is higher than 90% and lower than 110% of the estimate of most likely effort for each task in T. Assume that the mean value of his probability assessments,

AvConfLev(Int1,T), for these tasks is 50%. Assume further that the proportion of actual effort values included in the Int1 effort prediction intervals for the same set of tasks is only 30%, i.e., the HitRate(Int1,T) is 30%. Then the level of overconfidence is calculated as: Overconfidence(Int1,T) = 50% - 30% = 20%.

There are no standard measures of the performance of uncertainty assessment. In an earlier paper [7], 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. While Overconfidence(Int,T) measure ability to assess the average level of uncertainty, we introduce ***RecUncAbility(Int,T)*** to measure a developer's ability to assess relative difference of uncertainty between different tasks. RecUncAbility(Int,T) is defined as the correlation between the confidence in including the actual effort in Int and the estimation error (MRE) for the set of tasks (T). 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 and the estimation error. We expected therefore that these measures would yield high negative values if the estimator were skilled at assessing the relative difference in effort estimation uncertainty between different tasks.

4.3. Developers, tasks and experiment process of Study 1

We advertised at the University of Oslo for students who were highly-skilled Java program developers. We selected the five developers whom we believed to be the best Java programmers, based on examination of their *curricula vitae* and interviews. All developers had programmed several thousands lines of code in Java, had estimated the effort of similar tasks before, and had several months of programming experience in industry. For the goal of our study, these developers were considered to have the required competence. The student developers were paid for their work, which was not part of any coursework at the university.

The programming tasks assigned in the experiment were typical of those usually undertaken by the developers in their university context, so they had substantial experience with this type of task. The programming tasks were quite small. There were 18 different tasks in all. We included tasks that were independent of each other and not part of a larger software system, because we wanted to ensure that any increase in the accuracy of assessing estimation uncertainty was the result of learning and not facilitated by a better understanding of the software system. 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 a GUI or other types of visual solution, while others were text-based. The tasks were similar in type and complexity. The sequence of tasks was similar, but not identical, for all developers and there was no obvious difference in the tasks' complexity. The independence of the tasks from each other was investigated by consulting the MRE and RE after the study was finished. We found no improved accuracy of estimates of most likely effort. This indicates that there was no learning from task to task with respect to programming skill, which provides support for the claim that the tasks were independent of each other.

An example of a task is the following:

Task description for "Juke box": *Write a program to select an audio file using the file dialogue 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 computer laboratories at the Department of Informatics, University of Oslo. The specifications of all programming tasks are available by request to the authors.

The experiment took place over a period of about two weeks. Each developer participated for five work-days. On the first day of the experiment, all developers were informed that an important purpose of the experiment was

to study how well they were able to improve their effort uncertainty assessments. The developers had received the instructions for the experiment two days before it began. The requirement specifications for each programming task were handed out once the previous task had been estimated and then completed. Upon receiving a programming task the developer read the text, briefly analyzed the problem, and estimated the most likely effort (EstML). The estimated most likely effort was given as input to an HTML form. A PHP script 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. The developers chose from a finite set of probability values: 0, 5, 20, 35, 50, 65, 80, 95, and, 100%. To facilitate the interpretation we included an explanation of the probabilities that appealed to actual frequencies, such that a probability of x% will manifest itself, in the long run, in a frequency of x% interval hits. The reason for this explanation by appeal to long-term frequency is that Gigerenzer [30] found that this may reduce overconfidence in uncertainty assessments.

When working on a task the developers could take as many breaks as they liked. The time spent on breaks was not included in the actual effort. When the developers 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 in any way, the developers were asked to change or correct the program. Once the task had been completed satisfactorily, the developers 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. When all tasks had been completed, we interviewed the developers about their learning strategies and other issues relevant for understanding the results.

4.4. Developers, tasks, material and experiment process of Study 2

Ten professional consultants with the required experience for estimating and solving specified software development tasks were recruited through contact with software development consultant companies. The knowledge about development tools and programming languages we required dictated that we recruited only highly skilled, senior software developers. The software developers were paid standard industry fees. The developers knew that they were part of a study, but were instructed to behave as if this was ordinary consultancy work with us as clients. Their responses when interviewed after the tasks had been completed suggest that they estimated and programmed as they would have done in ordinary work situations. All of them had previous experience with uncertainty assessment of effort estimates.

All ten developers completed, and estimated the most likely effort of, the same five programming tasks in the same sequence. The programming time spent by the developers to complete these tasks varied from 11.5 to 45.5 work-hours. The programming tasks were maintenance tasks on an existing Java-based web-system with information about research papers. Short versions of the task descriptions are provided in Table 1. We assessed the first two tasks to be less complex than the last three. This increase in complexity meant that the *perceived* complexity of the tasks was not much different, and this may, to some extent, have compensated for the inexperience of the developers when starting to maintain what was, for them, a new software system. The strategy of starting with the easier tasks is an intuitively appealing, and commonly used, strategy when maintaining a new system. There were fewer tasks in Study 2 than in Study 1 because the tasks were larger and using software professionals instead of students as participants increased costs.

Table 1: Task descriptions (short versions)

Task 1	Add functionality that enables the user's last search to appear in the search text area at next log in. The search should be performed automatically at log in.
Task 2	Add support for the unused XML-tag "alternative_title". Add a field that is searchable, such that it is possible to sort the information and such that information is shown (a) in the search results if ticked off in the display settings and (b) in the information field when viewing a particular publication.
Task 3	Add functionality such that an administrative user is able to add new codes and categories used to classify publications. Up to now, the codes and categories have been hard-coded in the system.
Task 4	Add caching of the statistical data for when all publications in the system are viewed.
Task 5	Expand the functionality in Task 3 to enable the codes and categories used to classify publications to be added and deleted.

The main steps of a software developer's work were as follows:

1) When arriving, the developer was informed about the conditions for his employment, i.e. that this was a scientific study *as well as* an actual programming job they were hired to do. Then, he (all of them were male) was informed about the work process to be followed. He was informed that the total work load would be about one week's work, but not how many tasks he was supposed to complete.
2) The developer received the specification of Task 1 and the request to perform an "impact analysis", i.e., a brief analysis of the foreseen consequences of the task on the software system. The typical (median) time spent on the impact analysis was about 1 work-hour per task. The impact analysis was included to ensure that the effort estimation and uncertainty assessment work was of high quality and based on a good understanding of the work that had to be done.
3) The developer estimated the most likely effort and assessed the confidence he had that the actual effort would be inside Int1, Int2, and Int3, i.e., he assessed the uncertainty of his effort estimate. Then, he wrote a brief report on the processes used when estimating the effort and assessing the uncertainty of the effort estimate.
4) The developer completed the programming task in accordance with the requirements specification. We, in our role as clients, immediately tested the solution. If the solution did not pass our test criteria for acceptance, the developer had to improve the solution. This was repeated until the solution passed all our test criteria for acceptance.
5) The developer reported the actual effort used on a time sheet, i.e., he generated the outcome feedback he could use to evaluate the accuracy of the effort estimate and the realism of the uncertainty assessments. We did not explicitly instruct the developer to use the outcome feedback in producing the effort estimates and uncertainty assessment, but the information was readily available.
6) The developer received the specification for Task 2 and repeated steps 2)-5) for this task, then received the specification for Task 3, etc., until Task 5 was completed.
7) When all five tasks had been completed, we interviewed the developer regarding reasons for accurate and inaccurate estimates and uncertainty assessments, estimation and uncertainty assessment processes applied and learning processes.

5. Results Study 1

In order to ensure that a task would be completed within a working day, the sequence of the tasks was slightly different among the developers. This made 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 developers, because there were no content-related dependencies between the tasks. The developers solved

between 14 and 18 tasks during the five work-days of participation. The analysis is based on the same twelve tasks solved by all five developers. None of the developers deviated with more than two tasks from the originally planned sequence regarding these twelve tasks.

5.1. Development and estimation skill

Table 2 shows the total effort (TotEff) in work-hours needed to complete the tasks, the median absolute estimation error in % of actual effort (Median MRE), the median estimation error in % of actual effort (Median RE), the difference between the total actual effort and the total estimated effort (TotEff – TotEst), and the average time in minutes spent estimating a task (AvEstTime).

Table 2 Comparison of performance on the 12 tasks completed by all developers

Developer	TotEff (work-hours)	Median MRE	Median RE	TotEff – TotEst	AvEstTime
A	16:48	34 %	-31%	-2:52	6.8
B	19:36	18 %	1%	2:51	5.3
C	27:08	35 %	-2%	4:18	4.8
D	22:49	37 %	7%	2:24	5.4
E	24:49	22 %	-4%	-1:16	n.a.

Table 2 indicates that there were some differences in development and estimation accuracy. The median RE and the TotEff-TotEst values indicate that none of the developers were strongly overoptimistic about the effort required to solve the task. Developer A was in fact quite overpessimistic, particularly on the first tasks. Lack of overoptimism in situations with small tasks is not unusual and in accordance with results from other studies; see for example [31]. 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 judgmental estimation processes ("expert estimation"), sometimes supported by decomposition of the task into subtasks, to estimate the most likely use of effort. This is the estimation method most commonly applied in the software industry [32].

5.2. The ability to assess the average level of uncertainty

The high level of overconfidence in the accuracy of own effort estimates is shown in Table 3. The developers seem, on average, to be just as or even more overconfident in the accuracy of the last five as in the first five effort estimates. Notice that the lack of improved hit rates is a result of the design of the study, in that the independence of the tasks implies that the accuracy of the effort estimates is not likely to improve as the number of tasks completed increases.

Table 3: Hit rate and confidence level for the first and last five tasks

Measure	**Int1**	**Int2**	**Int3**
Hit rate first five tasks	16%	68%	92%
Confidence first five tasks	50%	82%	96%
Overconfidence first five tasks	34%	14%	4%
Hit rate last five tasks	8%	64%	88%
Confidence last five tasks	43%	80%	96%
Overconfidence last five tasks	35%	16%	8%

The observations in Table 3 are based on average values and may obscure the fact that some of the developers became more overconfident and some more realistic (i.e. may have learned from the feedback). For the purpose of analyzing the individuals' uncertainty assessment performance, we created a "learning graph" for each of the five developers. The learning graph is a useful tool for analyzing how well an individual developer was able to use the outcome feedback to adjust his assessment of the average level of effort uncertainty to the actual uncertainty. The learning graphs were 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}.

…

i) The comparison at Comparison Point *i* is conducted for T = {Task*i*…Task*i+4*}

…

Until Task*i+4* is the last task completed by the developer.

The rationale for including only the last five tasks in the set of tasks (T) is that it turned out to be a useful number for studying the learning effect. Including fewer tasks would lead to more random variation in the learning curve due to difference in task complexity, while including more tasks may have hidden some of the information about learning progress. Figure 1-5 depicts the "learning graphs" of developers A, B, C, D and E.

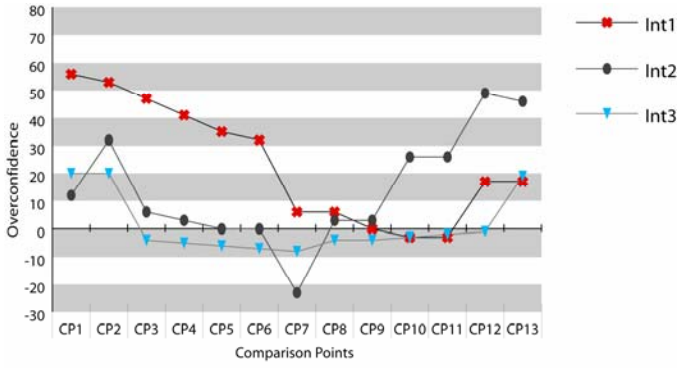

Figure 1 Learning graph of Developer A

Initially, Developer A was strongly overconfident, especially with respect to Int1, but then improved for both Int1 and Int3 and soon reached a good accordance between assessed and actual effort uncertainty. However, the uncertainty assessments related to Int2 went from overconfidence, to underconfidence, and then back to a higher overconfidence than in the beginning.

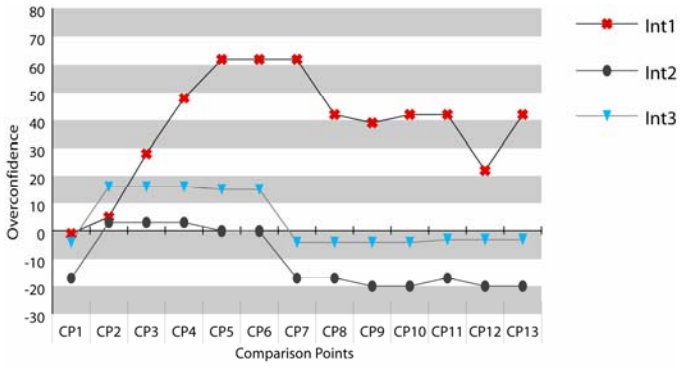

Figure 2 Learning graph of Developer B

Developer B had realistic assessments of the probability of including the actual effort in Int2 and Int3 for all comparison points. However, the developer went 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 than most of the other developers.

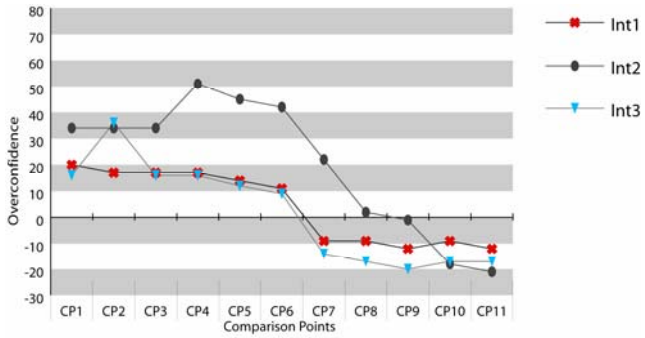

Figure 3 Learning graph of Developer C

Developer C went from overconfidence to underconfidence for all of Int1, Int2, and Int3. However, the level of underconfidence is not large and there are, we believe, clear signs of learning for all intervals.

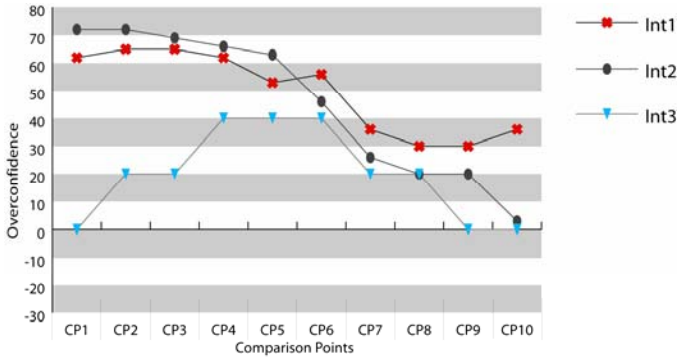

Figure 4 Learning graph of Developer D

Developer D started with a high degree of overconfidence for 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 all 15 tasks had been completed, there was a strong level of overconfidence regarding Int1. However, there are, we believe, clear signs of learning.

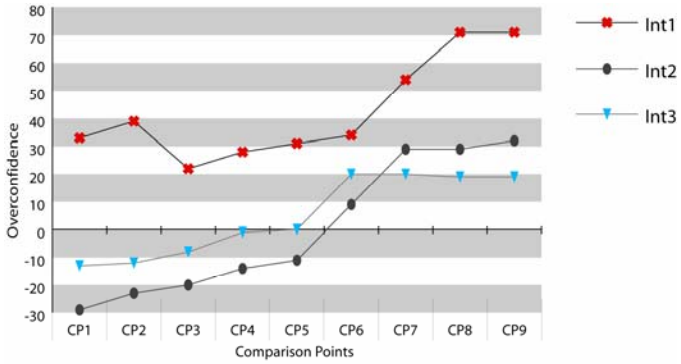

Figure 5 Learning graph of Developer E

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

There are, of course, many possible interpretations of the graphs in relation to learning. Much more in-depth evidence would be required to provide strong evidence supporting our interpretations. The graphs should consequently mainly be interpreted as weak evidence in support of a considerable variance in learning. While the performance of developers A, C, and D suggests learning from experience for at least one of the intervals, there was less sign of learning for developers B and E.

5.3. Ability to assess relative difference in uncertainty

Table 4 shows the correlation between confidence level and estimation accuracy for all developers and all tasks completed by the developers, applying the RecUncAbility measure, i.e., the correlation between level of confidence and estimation accuracy. Notice that there should be a strong negative correlation if the developers are good at distinguishing between high- and low uncertainty in effort usage 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 4 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 4, most of the correlations are low and/or positive, which suggest a poor ability to assess the relative differences in uncertainty between the tasks. However, the number of observations per correlation tests is low and we recommend a careful interpretation of the correlations.

A possible reason for this seemingly 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 significant correlation of -0.26 between confidence level and

estimation error. To determine whether there had been any learning related to the ability to assess the relative difference in estimation uncertainty, we compared the RecUncAbility (the correlations) of the first and last half of tasks of each developer. The comparison gave that Developers A, B and D were the only developers where the correlation between a high confidence in the estimation accuracy and low estimation error increased for the last half of tasks for at least two of the intervals, i.e., Developers A, B and D were the only developers who potentially improved their ability to identify tasks where the effort estimate was likely to be more than average inaccurate.

5.4. The uncertainty assessment strategies

The strategies used by the developers for uncertainty assessments and learning described in this section are based on (i) the comments provided by the developers after the completion of each task, (ii) the interviews with the developers after all tasks had been completed, and (iii) a comparison of the comments and interview information with the actual performance and uncertainty assessments. The amount of verbal information and analysis is high and the word limit for this paper allows only a brief summary of selected topics. The analyses were conducted independently by the two authors. There were only a few, minor, disagreements and these were resolved through subsequent discussion and further analysis.

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 lay inside the Int1 effort interval, then the confidence level for 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) and use of the estimation error of those tasks to set the confidence levels. For example, if the estimation error of a similar task 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., basically a strategy that is based on what "feels right".

Table 5 summarizes our categorization of the strategies applied by the different developers, according to information source.

Table 5 Main uncertainty assessment strategies

Developer	Sources: 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

Table 5 suggests a good match between the learning, as indicated in the learning graphs, and the use of uncertainty assessment strategy. The developers with the best learning from experience (developer A, C, and D) all applied a dominantly explicit uncertainty assessment strategy, i.e., strategy S1 or S2. The categorization of a strategy was sometimes difficult and had to be derived from written comments, interviews and actual behavior. When in doubt, we emphasized the actual behavior, e.g., on whether or no high errors on the previous tasks actually did lead to lower confidence on the subsequent tasks. Examples of our analysis are provided below:

- Comment by Developer A: "*I think I should increase the probability to include the actual effort in Int3 to nearly 100%, and about 90% for Int2, since I am seldom outside these intervals.*" This and similar

comments, together with how Developer A adjusted his confidence level from task to task, indicated that it was likely that he had summarized the hit rate of earlier uncertainty estimates and used this for his uncertainty assessments, i.e., that he had used strategy S1. The interview confirmed that this strategy had been used to some extent.

- Comment by Developer D: *"When the first estimate and uncertainty assessment was too optimistic, I learned that I should reduce my level of confidence.*" This statement, together with the actual adjustments conducted by Developer D on several tasks, led us to assess his strategy as typically based on the performance of one or two recently performed, similar tasks, especially when the estimation accuracy had been poor. We found few indications of summaries of hit rates of many previous tasks in his comments or in his actual uncertainty assessments, and chose not to categorize his strategy as S1, but as S2.
- Comment by Developer E: "*I believe the estimate was quite accurate on previous tasks and felt quite confident that the accuracy would be high on future tasks as well.*" Developer E used the word "felt" frequently and selected confidence levels that were not affected by the outcome feedback of previous effort estimates in any way we were able to discover. This led us to categorize his strategies as mainly S3, in spite of his reference to previous tasks in his statement.

The examples show that the differences in use of strategy may not be as clear as is suggested by Table 5. For example, all developers had elements of intuition-based strategies (S3) and it was not always easy to separate the use of S1 and S2. However, despite such problems of categorization, clear differences in uncertainty assessment strategies did emerge among the developers, and our categorization in Table 5 points to important elements of these differences.

The developers' own description of their use of uncertainty assessment strategy 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 or in his task comments. 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, or made some other mistakes. 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 reasons to believe that our categorization of strategy on the basis of the task comments and the analysis of chosen confidence levels is closer to the one actually applied than the one claimed in the interviews.

The interviews uncovered 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 interval boundaries it was nevertheless considered a hit. For example, actual effort only five minutes outside the maximum effort was usually interpreted to be a hit. In addition, very large estimation overruns was frequently considered to be outliers, i.e., not relevant as lessons learned for future assessments.

6. Results Study 2

In Study 1 the uncertainty assessment learning was examined in a context where students completed many tasks of similar type and where each task was independent of each other. A perhaps more common context is, however, that there are fewer tasks of similar type to learn from, that the tasks are related and that the tasks are completed by software professionals. The latter context is the basis for the Study 2 results reported in this section.

6.1. Development and estimation skill

Table 6 shows the total effort (TotEff) in work-hours needed to complete the tasks, the median absolute estimation error in % of actual effort (Median MRE), the median estimation error in % of actual effort (Median

RE), the difference between the total actual effort and the total estimated effort (TotEff – TotEst), and the average time in minutes spent estimating a task (AvEstTime).

Table 6 Comparison of work and estimation performance

Developer	TotEff	Median MRE	Median RE	TotEff – TotEst	AvEstTime
A	45:25	44%	44%	5:00	20
B	42:34	45%	36%	25:04	24
C	16:20	33%	27%	2:20	10
D	35:00	36%	35%	15:00	19
E	11:35	8%	3%	1:50	10
F	40:24	43%	7%	1:24	15
G	25:14	44%	-31%	6:14	9
H	17:51	27%	-9%	-2:54	13
I	29:27	63%	57%	12:17	11
J	30:51	22%	3%	4:36	25

Table 6 indicates that there are some differences in development skill, i.e., while the least productive developer used about 45 work-hours to solve the five tasks, the most productive used only about 11 work-hours. This is, we believe, not an unusually high productivity variance among software professionals. In fact, much higher productivity variances is typically observed in real life situations. As an illustration, [33] report a study of productivity variance of professional developers maintaining the same application and concludes that: "*Some programmers are better than others by an exceptionally wide margin. The best-producing people can be 20 or more times better than the low-end group. A factor of more than 100: 1 may separate programmers at the ends of the spectrum.*" Similarly, the estimation errors observed here are not uncommon and similar to the average rates of the surveys reviewed in [1].

6.2. The ability to assess the average level of uncertainty

Table 7 shows the level of overconfidence for all three uncertainty intervals and reports that the level of overconfidence fell from Tasks 1-3 to Tasks 4-5 for all three uncertainty intervals. The first impression when looking at Table 7 may be that the developers learned how to provide more realistic uncertainty assessments. Unfortunately, there are reasons to believe that the improvement regarding levels of overconfidence is not related to learning how to make better uncertainty assessments, but instead only to a better understanding of the maintained system, with more accurate estimates being a corollary. The main argument for this is that the mean confidence levels are, in spite of strong initial overconfidence, the same for Tasks 1-3 and Tasks 4-5. If the software developers had reflected properly about the realism of their confidence levels, we should have observed a decrease of mean confidence levels for Tasks 4-5. The strong decrease in estimation error (mean MRE) from 74% on the first three tasks to 28% on the last two tasks may, consequently, be the main explanation for the observed decrease in general level of overconfidence, and not improvement in the processes of assessing uncertainty. Alternatively, the unchanged confidence levels may have been a result of the software developers believing that their experience from the previous tasks had taught them how to avoid large estimation errors. In other words, the developers may have accepted that they had been overconfident on previous tasks, but believed that their previous confidence levels were now realistic due to learning. The interdependency of tasks in Study 2, which increased the opportunity to improve the estimation accuracy compared to more independent tasks in Study 1, may consequently easily harm the rather than supported the uncertainty assessment learning. In situations with more learning opportunities, over-estimation of actual learning may be even more frequent, which in turn may lead not only to unchanged, but increased, levels of overconfidence.

The collected data in Studies 1 and 2 do not enable us to explain why the software developers remained overconfident in spite of outcome feedback. There have, however, been many other studies on this topic that may contribute to such explanations. A review of the explanations presented in those studies is included in Section 7.3.

Table 7: Hit rate and confidence level for Tasks 1-3 and 4-5

Measure	Int1	Int2	Int3
Hit rate Tasks 1-3	7%	50%	60%
Confidence Tasks 1-3	56%	80%	94%
Overconfidence Tasks 1-3	49%	30%	34%
Hit rate Tasks 4-5	35%	70%	80%
Confidence Tasks 4-5	62%	81%	91%
Overconfidence Tasks 4-5	27%	11%	11%

The lower number of tasks completed in Study 2 than in Study 1 meant that it would not be meaningful to create "learning graphs" for the ten developers. Instead, we indicated the degree of individual learning through the individual hit rates (of the five tasks) and confidence levels for each task. Tables 8-10 display the data for the uncertainty intervals Int1, Int2 and Int3, respectively. When analyzing the individual developers' level of learning, we assumed that 1) a developer's overconfidence (underconfidence) in the accuracy of his effort estimates is reflected in confidence levels higher (lower) than his overall hit rate, and 2) an overconfident (or underconfident) developer is probably not learning if he is not providing confidence levels closer to the overall hit rate on the last tasks than on the first tasks.

Table 8: Changes in confidence levels (Int1)

Developer	Hit rate	Conf. Task 1	Conf. Task 2	Conf. Task 3	Conf. Task 4	Conf. Task 5
A	0%	70%	80%	70%	80%	70%
B	0%	60%	60%	70%	70%	80%
C	20%	30%	40%	10%	40%	25%
D	0%	80%	75%	80%	70%	70%
E	60%	50%	70%	70%	70%	50%
F	40%	20%	10%	10%	5%	10%
G	0%	50%	60%	60%	80%	70%
H	40%	60%	75%	80%	75%	70%
I	0%	60%	70%	80%	70%	85%
J	20%	40%	40%	60%	60%	75%

Applying these two assumptions on the data in Table 8, we find that the developers A, B, D, G, I and J are strongly overconfident and we find no indications of learning related to Int1. Developer H is weakly overconfident and developer F is weakly underconfident, but neither of them show indications of learning, either. Developers C and E were reasonably well calibrated from the beginning and may have learned successfully at earlier stages.

Table 9: Changes in confidence levels (Int2)

Developer	Hit rate	Conf. Task 1	Conf. Task 2	Conf. Task 3	Conf. Task 4	Conf. Task 5
A	40	80	90	80	90	80
B	40	80	80	80	80	90
C	80	80	90	80	90	90
D	40	85	85	90	80	80
E	80	90	80	90	80	80
F	40	75	70	70	40	50
G	80	75	80	70	90	90
H	80	75	85	90	85	90
I	20	70	80	90	80	90
J	80	60	75	80	70	85

In relation to Int2, developers A, B, D, and I are strongly overconfident and show no indications of learning. None of the developers are underconfident. Developers C, E, G, H and J were well calibrated from the beginning. Interestingly, developer F seems to have started rather overconfidently and then adjusted the level of confidence towards the hit rate, i.e., there are indications of successful learning by developer F.

Table 10: Changes in confidence levels (Int3)

Developer	Hit rate	Conf. Task 1	Conf. Task 2	Conf. Task 3	Conf. Task 4	Conf. Task 5
A	60	95	95	97	95	95
B	60	99	99	99	99	99
C	80	99	99	99	99	99
D	60	95	95	95	90	90
E	80	100	95	99	90	95
F	80	90	80	80	50	70
G	80	100	90	90	100	100
H	80	90	95	95	95	99
I	20	90	90	95	90	95
J	80	99	80	90	80	95

In spite of the very wide effort interval represented by Int3, none of the developers had a hit rate of 100%. Developers A, B, C, D, E, G, H, I and J were all, to some extent, overconfident and showed no indications of learning. Similarly to Int2, the only observable attempt to adjust the confidence level to the hit rate was made by Developer F.

Developer F was neither one of the most skilled developers, nor had he the most accurate effort estimates. However, he did have one of the most unbiased (median RE = 0.07) effort estimates, i.e., he was, on average, neither overoptimistic nor overpessimistic. This supports the impression that developer F had the ability to learn the average level of effort required and the average level of uncertainty of the type of tasks solved from outcome feedback. In the interviews, Developer F stated that he had learned from the outcome feedback, e.g., that after a while he "*better understood the uncertainty of estimates*" and became "*more humble*" about his own ability to estimate the effort accurately.

Developers C and E were well calibrated with respect to Int1 and Int2 from the beginning, which may indicate that, at some earlier stage, they had successfully improved their ability to assess the effort estimation uncertainty of the type of tasks completed. These developers were the two most skilled developers, i.e., they expended the least amount of effort on completing the five tasks. Hence, the most important reason for their

reasonable accurate uncertainty assessments may be the lack of unexpected problems as a result of their high programming skill, and not feedback-based learning from previous tasks. Nevertheless, they remain at least candidates for successful learners regarding uncertainty assessment.

6.3. Ability to assess relative difference in uncertainty

Table 11 shows the correlation between confidence level and estimation accuracy (MRE) of the five tasks for each developer, applying the RecUncAbility measure, i.e., the correlation between level of confidence and estimation accuracy. As before, there should be a strong negative correlation if the developers are good at distinguishing between tasks with a low and high likelihood of inaccurate effort estimates. A positive correlation means that it is more typical that effort estimates that are assessed to have low estimation errors in fact have high estimation errors, and vice versa.

Table 11 Correlation between confidence level and MRE

Developer	Int1	Int2	Int3
A	0,2	0,2	0,8
B	0,7	0,8	*
C	0,3	0,3	*
D	0,2	0,5	0,1
E	0,6	0,1	0,3
F	0,0	0,4	0,3
G	-0,7	-1,0	-0,6
H	0,2	0,0	0,0
I	-0,1	-0,1	-0,2
J	0,8	0,8	0,2

* All confidence levels were equal

As can be seen from Table 11 most of the correlations are positive or low; not the result we would expect if the software developers were good at separating high and low uncertainty tasks. Developers G and I are the only ones with negative correlations on all three intervals.

A possible reason for this finding is the phenomenon “regression towards the mean” in combination with an over-reaction to the outcome feedback when having very accurate or inaccurate effort estimates. When examining the developers’ comments on the process of uncertainty assessment, together with their performance on the previous task, we observed that the chosen confidence levels were sometimes strongly affected by the estimation error on the task just completed. For example, suppose that the estimation error on Task 1 was high and that the estimation error of Task 2 was low. Then, the confidence in the estimate of Task 3 was similar to that in Task 2, despite there having been a high estimation error on Task 1. In general, a relatively low estimation error is more likely followed by a higher estimation error; see our analysis of the “regression towards the mean” in [34] for an explanation for this statistical phenomenon. A strong impact from the previous task’s level of estimation error on the level of confidence in the following effort estimate will therefore typically lead to a positive correlation between estimation error and confidence level and explain the results in Table 7. However, more studies are needed to examine the robustness and validity of this explanation; it may be that the pattern observed in our studies may not be typical in other contexts.

6.4. The uncertainty assessment strategies

The strategies for uncertainty assessment described in this section are based on (i) the process descriptions provided by the developers after each effort estimate and uncertainty assessment, and (ii) the interviews with the developers after all tasks had been completed. Similarly to the analysis in Study 1, the analyses were conducted independently by the two authors.

As in Study 1, we tried to categorize 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. *S2 (Analogy-based strategy)*: uncertainty assessment through recall of a small set of similar tasks and use of the estimation error of those tasks to set the confidence levels. *S3 (Intuition-based strategy)*: uncertainty assessment without any explicitly formulated strategy. See Section 5.4 for further description of these strategies. The authors had a few disagreements in categorizations related to S1 and S2. In hindsight, this is not surprising, due to the fact that the much shorter learning sequence in Study 2 did not allow a calculation of hit rate (use of S1) based on more than four tasks. For this reason, we decided to combine S1 and S2 into the category S12, i.e., the use of S1, S2 or both S1 and S2. The main difference between S12 and S3 is that S12 contains more explicit strategies than S3. The only other disagreement was related to the degree to which a developer should use a strategy to be classified as using it. To resolve this disagreement, we decided to include all strategies that could be derived from the tasks comments or interviews. Once these two changes had been implemented, there were no further disagreements in classification.

The following quotations from the developers' own descriptions of uncertainty assessment process exemplify what we looked for when interpreting a strategy as explicit (S12) and/or intuition-based (S3):

- S12 + S3: (Developer H): "*Gut feeling. The task seems quite straightforward, but based on the previous four tasks, which I have not quite been able to estimate very well, I place a relatively low probability figure (70%) on the 90-110% interval.*" The reason for classifying the strategy as, to some extent, explicit (S12) is that Developer H uses the performance of the previous task to adjust the uncertainty assessment of this task. The additional use of reference to "gut feeling" led to the categorization S12+S3.
- S3 (Developer J): "*I'm not 100% sure that my solution will work. I'm not sure the modifications will not have any unwanted effects, and I am uncertain if the cached data will be updated the way it should.*" Developer J describes his feelings and makes no reference to previous tasks or analytical uncertainty assessment strategies. The strategy was consequently categorized as S3.

Table 12 summarizes our categorization of the strategies applied by the different developers, according to information source.

Table 12 Uncertainty assessment strategies

Developer	Source: Task comments	Source: Interviews
A	S3	S3
B	S3	S3
C	S12 + S3	S3
D	S12 + S3	S3
E	S12 + S3	S12 + S3
F	S12 + S3	S12 + S3
G	S12 + S3	S12 + S3
H	S12 + S3	S12 + S3
I	S12 + S3	S3
J	S3	S3

Table 12 reflects the observation that developers C, D, E, F, G H and I applied explicit uncertainty assessment strategies (use of S12) in at least one of the tasks. Developers C, F, H, and I described the use of S12 on at least two tasks. Developers A, B, and J described the use of intuition-based uncertainty assessment strategies (use of S3) on all tasks.

For most developers, their own description in the interviews of what strategy they used for assessing uncertainty matched quite well the strategies we derived from the task comments for most developers. The developers were rather modest about their use of explicit strategies and willingly admitted that many of the assessments were based on what they felt was the correct level of confidence, i.e., they were based on intuition. The following quotation from the interview with Developer H, who was one of the most frequent users of explicit uncertainty assessment strategies, i.e., S12, illustrates this modesty: "*It is the gut feeling I have about the estimate that has the largest impact on my uncertainty assessment.*" This statement was followed by a description of how he tried to combine this gut feeling with the estimation performance on similar, previously completed tasks.

This willingness to admit the strong use of gut feeling separated the professional, to some extent, from the student developers in Study 1. The student developers seem to have felt more strongly that they should not use intuition-based uncertainty assessment strategies. The student developer E in Study 1, for example, showed no evidence of using analytical uncertainty assessment strategies, but stated nevertheless that he had been: "*Most analytic. It has always been like that. The gut feeling is always there, but it does not determine the final assessment. I analyze the information I have.* [The uncertainty assessment is] *About 90% based on analytic thinking and 10% gut feeling.*" By contrast, most of the professional developers had much less of a problem in using intuition as an important part in their assessments.

7. Discussion of results

7.1. Design rationale and threats to validity

Generalization

It seems to be common in software engineering to assume that generalization from a random sample of a population to the population itself should be the means by which experimental results should be generalized [35]. This assumption has as a consequence that artificiality is automatically seen as a threat to validity. Consequently, our use of a learning friendly environment and small programming tasks may easily be seen as a threat to the validity of the study. The problem with the above assumption is that it does not recognize that there are several other legitimate ways of making generalizations and that increased artificiality can be used as a deliberate instrument for acquiring knowledge. The potential benefits of artificiality and the variety of means to generalize are widely recognized outside the software engineering research community. Webster [36], for example, argues that when studying group processes, "*experiments must become more, not less, artificial to be most useful in understanding everyday group processes*". Shadish et al. [37] provides five principles for generalizing in the absence of random sampling from a well-defined population: surface similarity, ruling out irrelevancies, making discriminations, interpolation and extrapolation, and causal explanation.

If we had intended to generalize from a representative sample to a larger population of software professionals through statistical means or to generalize by similarity, our decision to use small tasks and in-depth studies of only five software engineering students and ten software professionals would have been difficult to defend. However, that was not our intention. We did not expect or intend that our results would be transferable directly to a typical software engineering context, in which professionals build large software systems. Our study tries instead to generalize *across populations* through causal explanation and analytical reasoning. For such cross-population generalization, the use of small programming tasks was a deliberate part of the study design and was not a consequence of poor design or lack of access to larger tasks. Larger, more 'realistic', tasks would increase

the time from prediction to feedback, introduce impact from more nonstudied factors and make the analysis of learning difficult. It would, for example, be difficult to know whether a lack of learning was the result of the long wait from estimation to feedback or the uncertainty assessment strategy applied. We believe that the deliberately introduced artificiality elements of our studies are useful elements to enable an examination of important conditions for realism and learning in effort estimation uncertainty assessment.

Learning motivation

The main threat to the results of our study is, we believe, related to the developers' motivation for learning regarding uncertainty assessment. The major concern of software developers is, typically, to possess and improve software design and programming skills. The assessment of effort estimation uncertainty may be perceived as a secondary, less important skill. A low priority assigned to assessing effort estimation uncertainty on the part of developers, may imply that the motivation to learn how to improve their assessments will be small; yet a prerequisite for learning is, precisely, strong motivation. With this difficulty in mind, we took several precautions to ensure that the developers would possess the requisite motivation during the study. For example, the software developers had to describe the process they used to assess the uncertainty after each task and they knew that we would analyze the realism of their uncertainty assessments and they were paid ordinary salary for their work. The motivation for learning may nevertheless have been lower than was desired and we cannot claim to measure their potential of learning in situations in which their motivation to learn may be higher. There is, on the other hand, no reason to believe that the motivation for learning uncertainty assessment skills during the studies was any *lower* than in more realistic contexts for software development. In software development projects, the explicitly stated delivery and learning goals do not, in our experience, typically include uncertainty assessment learning. This means that it is even less likely to observe more uncertainty assessment learning from outcome feedback that we did in more realistic situations. A lower than desired motivation does consequently not limit our ability to generalize to typical software development situations. It does, however, leave the issue of what would happen if we succeeded in motivating the developers to think of uncertainty assessment learning as a primary goal open.

In both studies, the developers were aware of the general goals of the studies, e.g., they knew that we wanted to study learning and accuracy of uncertainty assessments. Our impression from interviews with the developers is that this knowledge had little, if any, impact on the learning strategies. If there was any effect on how the developers worked, it is likely that this was in the direction of more explicit strategies for assessing uncertainty, given that we had requested a description of them, i.e., that we would observe more rather than less learning in comparison with typical real-life situations.

Isolation of studied factors

In our two studies we tried to isolate the effect of outcome feedback and uncertainty assessment strategy on realism and learning. However, the high number of potential alternative reasons and the few developers observed means that there is a possibility that the relationships we observed are not causal, e.g., that the strategy for assessing uncertainty merely correlates with the underlying causal variable. This is a problem in most observational studies, particularly when there are few observations, and can only be solved through extensive data collection, analysis and proper discussions of results in relation to previous work. It emphasizes, however, that our results should be interpreted carefully.

Theory-laden observations

We, as researchers, may easily, unconsciously or consciously, start to adjust the analysis of uncertainty assessment learning strategy to fit our theories about how the strategy for assessing uncertainty affects the actual learning. To avoid this type of bias, we ensured that one of the authors had no information about the uncertainty assessment performance when analyzing and categorizing the strategies.

Male-dominated population

The population studied was composed of male students and software professionals, i.e., it does not include female developers. Previous studies suggest that females in general have slightly less overconfident assessments of own abilities than men [38]. This suggests that inclusion of female developers could have led to slightly better learning from outcome feedback. We have found no studies on effort estimation uncertainty assessment in relation to gender.

Individual vs team-based learning

We have studied individual learning and not learning in teams. This is a deliberate design choice, but limits our ability to generalize the results. Studies on teams' level of overconfidence typically, but with many exceptions, tend to report that teams are more likely to make overconfident decisions [39]. Our observations in software effort estimation uncertainty assessment situations, however, suggest that teams are less overconfident [40], possibly because they are more willing or able to use historical data. It is consequently possible that we would observe better uncertainty assessment learning when applying teams as study unit. More studies are needed on the effect of appropriate team learning processes on uncertainty assessment learning.

Tasks vs projects

Our studies examined the assessment of uncertainty regarding the effort estimation of programming tasks and not of projects. While the level of overconfidence in the accuracy of effort estimates of projects and of smaller tasks seem to be similar, it is not obvious that the learning processes and abilities are the same. While there is a need for more studies on this topic, we believe that it will be hard to improve uncertainty assessments on a project level without the ability to do so at a task level. If this belief is valid, our results are relevant for the ability to learn from outcome feedback in the context of large projects, as well as for smaller tasks.

7.2. Comparison of Study 1 and Study 2

The differences in study designs in Studies 1 and 2 enable us to make several potentially interesting comparisons:

- The student developers had completed the same education and had similar working experience; hence, their background and experience varied less than the software professionals. The use of student developers, as in Study 1, instead of software professionals, as in Study 2, had therefore as consequence that the homogeneity of the subjects' responses increased. An increased homogeneity on non-studied factors is in many situations an advantage, e.g., it may be easier to isolate the studied factor and the power of the tests may increase. When the effect sizes are small, this advantage of use of student developers may sometimes be essential for the quality of a study.
- While the effort estimates of the student developers in Study 1 were typically unbiased or even over-pessimistic, the software professionals of Study 2 had a strong tendency towards over-optimism; all but two of the software professionals had a positive median RE. This, together with the evidence presented in Section 3, supports the hypothesis that over-optimism when estimating one's own work is not something that is eliminated with more on-the-job practice. It may instead increase.
- In both studies, the developers applied similar judgmental estimation and uncertainty assessment processes. Consequently, there were no visible differences in the general approach to estimation and uncertainty assessment used by students and software professionals. This suggest that the estimation and uncertainty assessment processes of student developers may represent those of the software professionals well, although we cannot exclude that the unconscious, mental steps involved in the judgments may be different, due to differences in types and amount of experience. Differences between students and software professionals may

however be greater when estimating the effort and assessing the effort uncertainty of software projects instead of programming tasks.

- The consistency between the results in Study 1 and Study 2 on many issues, in spite of the differences in design, increases the robustness of the main results. This includes in particular the result that software developers are poor at separating accurate and inaccurate effort estimates and that outcome feedback is frequently not sufficient for improvement of assessments of effort estimation uncertainty.
- A simple uncertainty assessment model would be to base the confidence level on the hit rate of the set of previous effort uncertainty assessments. An informal analysis suggests that a replacement of the judgments with this simple model would on average improve the uncertainty assessments in Study 1, but not in Study 2 where there were fewer, dependent tasks. This illustrates the point we made in Section 3, i.e., that the benefits from use of formal uncertainty assessment models as opposed to informal judgmental processes depends on the amount of historical data and the dependency of the tasks.

7.3. The importance of learning strategy

It is a difficult task to asses the uncertainty of software development effort estimates. In our study, we designed the task-solving environment and the feedback in a way that we believed would be more learning-friendly than typical real-life environments. Elements that we expected would generate a learning-friendly environment included the assignment of many similar tasks and the provision of immediate, easily accessible feedback. In spite of this, most developers did not improve their uncertainty assessment on the basis of the outcome feedback and remained strongly over-confident about the accuracy of their own effort estimates, even after having received extensive amounts of feedback.

It seems that the strategies of the developers with the best learning and the most realistic uncertainty assessments typically were more explicit and less intuition-based than those of the other developers. This suggests that the ability to learn depends, to some extent, on the use of an explicit strategy for uncertainty assessment, i.e., the use of S1 or S2. This is illustrated by the observation that of the six developers with the most realistic uncertainty assessments of the estimation accuracy of the final tasks and/or observable improvement from the first to the final tasks (developers A, C, D in Study 1, and developers C, E, F in Study 2), all but one (developer E in Study 2) seemed to have used explicit uncertainty assessment strategies. Of the remaining nine developers, all but two used a dominantly intuition-based strategy. Our assessment of strategy was conducted independently of the analysis of actual learning by one of the authors. That being so, we do not believe we have categorized the subjects' assessment strategies to fit our theory.

Our observation about the importance of using an explicit uncertainty assessment strategy for learning is supported by results from other domains, e.g., studies reporting that intuition-based assessments may be even more overconfident when a greater amount of information is available [11, 41], and that overconfidence is difficult to avoid when processes are intuition-based [42].

We do not claim that there actually will be learning from experience in typical software development situations when applying strategies S1 or S2. In [43], for example, we found that several professional estimation teams, even when they had collected information about their previous estimation performance, had a strong tendency to believe that their current effort estimate was accurate, even in situations where they knew that the estimates in similar situations had typically been very inaccurate. However, our results may indicate that learning and realism is more likely when using explicit uncertainty assessment strategies. It is, of course, possible that intuition-based strategies sometimes also lead to improved uncertainty assessment with more feedback, but as we will argued in Section 7.4, a continued over-confidence is more likely to be the result.

7.4. The persistence of overconfidence in own estimation accuracy

The observed persistence of overconfidence of many of the software developers in our studies is consistent with results in real-life software engineering settings; see related work in Section 3. To investigate the basis of the observed persistence in overconfidence, we conducted a search in the digital libraries INSPEC and PsychINFO (search conducted November 23, 2005) using the search term 'overoptimism OR overoptimistic OR overconfidence OR overconfident'. This returned 536 records, about 400 of which were relevant for our study. We reviewed the relevant studies with the goal of identifying obstacles to learning how to assess uncertainty.

The most striking observations from this review were, in our opinion, these: 1) the variety of domains and tasks regarding which people are overconfident about their own estimation ability, and 2) how seldom debiasing techniques, such as feedback about previous performance, helps when the judgments are intuition-based. On the basis of the review, we briefly summarize possible reasons for the persistence of overconfidence and why outcome feedback may not help in many cases. Our results are discussed relative to these reasons[3]. We developed the following four, partly overlapping, categories of possible reasons for persistent overconfidence:

- Biology-based reasons
- Culture-based reasons
- Motivational reasons
- Cognition-based reasons

Biology-based reasons: In [44] studies on the possible evolutionary basis of overconfidence are summarized. The author of that study terms the phenomenon 'adaptive overconfidence'. He argues that inflated confidence about one's own performance would bring numerous advantages in our evolutionary past and may have evolved as part of our adaptation to the environment. Furthermore, he argues that overconfidence about one's own strength has greater evolutionary benefit than, say, a conscious lie. For example, a conscious lie about one's own strength may be easier to discover by an opponent than an overconfident, unconscious claim about it. Consequently, an intuition-based learning strategy may have evolved to maintain, rather than change, a degree of over-confidence in one's own ability.

The actual benefits of overconfidence in today's world is also documented in many studies; see for example [45, 46]. A seemingly robust finding is that a high level of overconfidence is an indicator of above average well-being and health.

It is possible that the evolutionary basis of overconfidence is an important determinant for the problems that several of the developers in our study had in replacing intuition with more analysis-based uncertainty assessment strategies leading to more realism and perhaps even pessimism. A possible consequence is that developers need to be trained in applying analysis in uncertainty assessment to replace the intuition-based strategy that dominates in many situations.

Culture-based reasons: The individuals' responses take place in a cultural context and there are good reasons to believe that there is a relationship between culture and individuals' levels of overconfidence. There is, for example, evidence to suggest that Canadians are more overconfident than Japanese [47] due to a stronger focus on individual self-enhancement. However, results are inconsistent and in [48] it is reported that those in Asian cultures were more overconfident than those in Western cultures. The diverging results suggest that culture, including organizational culture [49], can affect the level of overconfidence, but that the cultural effect is not well understood at present.

Software development cultures vary in how they deal with overconfidence. The developers in our study may have been rewarded for overconfidence in previous software development work. This previous experience may,

[3] The full list of references from the review will be sent to interested readers on request to one of the authors.

perhaps unconsciously, have made it difficult to initiate a change from overconfident intuition-based assessments to analytic, realistic ones. Information on this topic is meager and further research is required.

Motivational reasons: Many studies show how people become overconfident about their own abilities in certain circumstances; for example, when they want to see themselves as skilled, when they believe that they will benefit from being perceived by other people as skilled, or when they believe that optimism will lead to better performance. For a comprehensive review, see [50]. In many contexts, including software development, this overconfidence may, to some extent, be rational. In [6], for example, we report that the managers assessed a developer to be more skilled when providing strongly overconfident effort prediction intervals than when he was being more realistic. This evaluation did not change when the manager was informed about the level of overconfidence. Overconfidence was not punished, but instead rewarded and used as an indicator of skill.

Software engineers, and particularly project managers, are typically motivated to think that they should be able to control events, e.g., that a skilled software developer should be able to plan and manage risk, and not be a victim of uncontrolled, unexpected events. This motivation is beneficial in many ways; for example, it may have a positive impact on performance and increase the likelihood that the desired end-state will be brought about [51]. The problem is that the overconfidence related to motivated reasoning is systematically higher than the increase in performance following an overconfident prediction; see for example [52].

These results may contribute to an explanation of the problems of learning from outcome feedback in our study. It is, for example, possible that a developer's possessing low confidence would be tantamount to admitting that he was not in control, which would imply a 'cognitive dissonance' between the skilled developer they believe and/or wish they were, and their actual assessments.

Cognition-based reasons: Cognitive reasons for overconfidence may have their basis in biological, cultural and/or motivational reasons. For example, the motivational-based tendency to overestimate one's own ability to control future events is also related to the cognitive tendency to attribute a positive outcome to one's own general ability to control events, and a negative outcome to specific external situations and bad luck [53]. Describing a reason as cognition-based, cultural, or motivational-based may frequently amount to a choice of the kind of explanation required.

Possibly the most interesting potential cognition-based reason for persistent overconfidence in effort estimation is the role of 'future focus'. When estimating effort, a software developer typically tries to foresee the necessary steps, break down the task into activities, and think about strategies to complete it successfully [22]. A future focus is necessary to plan and complete a project, but has repeatedly been shown to lead to overoptimistic effort estimates and overconfidence in the accuracy of the estimates [54]. By contrast, a stronger focus on looking back, or separating the processes of planning and estimating, is more likely to lead to more realistic estimates and less overconfidence [55].

An additional, interesting cognition-based reason for lack of learning is that people tend to overestimate what they have learned from previous experience [56]. A consequence of this bias is that analyzing previous experience may lead to increased overconfidence if the belief in what one has learned from past experience is not justified. There may be less to learn from past experience than we believe, and trying to learn from it may consequently result in an increase in, rather than a reduction of, the overconfidence, e.g., overconfidence in our ability to control events. Consequently, the developers in our study, particularly when applying intuition-based strategies, may have been affected negatively by the outcome feedback. This possibility of increasing the overconfidence with more outcome feedback emphasizes the need to apply strategies for uncertainty assessment that improve as more data is available, e.g., S1 and S2.

8. Conclusion

Our study tried to answer the following two research questions: 1) How much do software developers improve their assessments of the uncertainty of estimates of most likely effort on the basis of outcome-related

feedback?, and 2) What is the relation between the uncertainty assessment strategies used by the software developers and their ability to improve their uncertainty assessments?

To answer these questions, we conducted two studies. In Study 1, five student developers completed between 14 and 18 programming tasks each, with estimation accuracy feedback being provided after each task. In Study 2, 10 professional developers solved five programming tasks each, also with estimation accuracy feedback being provided after each task. We address the learning related research questions in the context of rather small development tasks, and, not in the context of larger software projects. If we observe poor uncertainty assessment learning in the addressed context with many tasks and immediate estimation accuracy feedback, there are reasons to believe that we will observe even poorer learning in a context with few, large projects and delayed feedback.

With respect to research question 1, we conclude that most developers found it difficult to assess the uncertainty of their estimates and remained strongly over-confident about their ability to estimate accurately, in spite of the estimation accuracy feedback. The software professionals in Study 2 did not perform any better, perhaps worse, than the student programmers in Study 1. Few of the developers improved their uncertainty assessment through the feedback. We assessed that three out of the five student developers in Study 1 improved, but perhaps only one out of the 10 professional developers in Study 2. The main reason for this difference in learning between Study 1 and 2 may be that there were fewer tasks to learn from in Study 2.

With respect to research question 2, we interpret the results from Studies 1 and 2 to support the hypothesis that an important condition for improvement based on outcome feedback is the use of an explicit strategy for assessing uncertainty that improves as greater amounts of historical data become available. This hypothesis is also supported by the results from our review of about 400 studies on overconfidence in various contexts. An implication of the results is that we cannot expect uncertainty assessments to improve much when they are dominantly intuition-based.

A possible implication of our results is that software companies should train and instruct software development effort estimators to use explicit strategies for uncertainty assessment. In particular, we recommend the use of the strategies S1 or the strategy described in [7].

References

1. Moløkken, K. and M. Jørgensen. *A review of software surveys on software effort estimation*. in *Proceedings of the 2003 International Symposium on Empirical Software Engineering*. 2003. Rome, Italy: IEEE Computer Society. p. 223-230.
2. Pietrasanta, A.M. *Current methodological research*. in *ACM National Conference (USA)*. 1968: ACM Press, New York, NY, USA. p. 341-346.
3. Li, H., Q.P. Shen, and P.E.D. Love, *Cost modelling of office buildings in Hong Kong: an exploratory study.* Facilities, 2005. **23**(9/10): p. 438-452.
4. Kitchenham, B., et al., *An empirical study of maintenance and development estimation accuracy.* Journal of Systems and Software, 2002. **64**(1): p. 57-77.
5. Shenhar, A.J., *One size does not fit all projects: Exploring classical contingency domains.* Management Science, 2001. **47**(3): p. 394-414.
6. 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, 2004. **70**(1-2): p. 79-93.
7. Jørgensen, M., *Realism in assessment of effort estimation uncertainty: It matters how you ask.* IEEE Transactions on Software Engineering, 2004. **30**(4): p. 209-217.
8. Connolly, T. and D. Dean, *Decomposed versus holistic estimates of effort required for software writing tasks.* Management Science, 1997. **43**(7): p. 1029-1045.
9. Einhorn, H.J. and R.M. Hogarth, *Confidence in judgment: Persistence of the illusion of validity.* Psychological Review, 1978. **85**(5): p. 395-416.

10. Klayman, J., et al., *Overconfidence: It depends on how, what and whom you ask.* Organizational Behaviour and Human Decision Processes, 1999. **79**(3): p. 216-247.
11. Oskamp, S., *Overconfidence in case-study judgments.* Journal of Consulting Psychology, 1965. **29**(3): p. 261 - 265.
12. 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): p. 123-146.
13. 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): p. 159-183.
14. Hammond, K.R., *Human judgement and social policy: Irreducible uncertainty, inevitable error, unavoidable injustice.* 1996, New York: Oxford University Press.
15. Hughes, R.T., *Expert judgement as an estimating method.* Information and Software Technology, 1996. **38**(2): p. 67-75.
16. Bolger, F. and G. Wright, *Assessing the quality of expert judgment: Issues and analysis.* Decision Support Systems, 1994. **11**(1): p. 1-24.
17. 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): p. 844-855.
18. Gagne, R.M. and K.L. Medsker, *The conditions of learning: training applications*. 1996, Fort Worth: Harcourt Brace.
19. Ericson, K.A. and R.T. Krampe, *The role of deliberate practice in the acquisition of expert performance.* Psychological review, 1993. **100**(3): p. 363-406.
20. Hogarth, R.M., *Educating intuition*. 2001, Chicago: University of Chicago Press.
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): p. 123-136.
22. Jørgensen, M., *Top-down and bottom-up expert estimation of software development effort.* Information and Software Technology, 2004. **46**(1): p. 3-16.
23. Moløkken, K. and M. Jørgensen, *Expert Estimation of Web-Development Projects: Are Software Professionals in Technical Roles More Optimistic Than Those in Non-Technical Roles?* Empirical Software Engineering, 2005. **10**(1): p. 7-30.
24. Jørgensen, M., B. Faugli, and T.M. Gruschke, *Characteristics of Software Engineers with Optimistic Predictions.* Journal of Systems and Software, To appear.
25. Jørgensen, M., *Evidence-Based Guidelines for Assessment of Software Development Cost Uncertainty* IEEE Transactions on Software Engineering, 2005. **31**(11): p. 924-954.
26. Humphrey, W.S., *The PSP and personal project estimating.* American Programmer, 1996. **9**(6): p. 2-15.
27. Prechelt, L. and B. Unger, *An experiment measuring the effects of personal software process (PSP) training.* IEEE Transactions on Software Engineering, 2000. **27**(5): p. 465-472.
28. Harrison, W., *Editorial (Based on this editorial, the paper "N=1: An alternative for software engineering research?" appeared in the ICSE Workshop: Beg, Borrow, or Steal: Using Multidisciplinary Approaches in Empirical Software Engineering Research, Limerick, Ireland, 2002).* Empirical Software Engineering, 1997. **2**(1): p. 7-10.
29. Zendler, A.E., et al., *Demonstrating the usage of single-case designs in experimental software engineering.* Information and Software Technology, 2001. **43**(12): p. 681-691.
30. Gigerenzer, G. and U. Hoffrage, *How to improve Bayesian reasoning without instruction: Frequency formats.* Psychological Review, 1995. **102**(4): p. 684 - 704.
31. Gray, A., S.G. MacDonell, and M.J. Shepperd. *Factors systematically associated with errors in subjective estimates of software development effort: the stability of expert judgment*. in *IEEE 6th International Metrics Symposium*. 1999. Boca Raton, Fl.: IEEE Computer Society. p. 216-227.
32. 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.
33. Bryan, G.E. *Not All Programmers Are Created Equal*. in *IEEE Aerospace Applications Conference Proceedings*. 1994. New York, USA. p. 55-62.
34. Jørgensen, M., U. Indahl, and D. Sjøberg, *Software Effort Estimation by Analogy and Regression Toward the Mean.* Journal of Systems and Software, 2003. **68**(3): p. 253-262.

35. Jørgensen, M. and D. Sjøberg. *Generalization and Theory-Building in Software Engineering Research*. in *Empirical Assessment in Software Engineering (EASE2004)*. 2003. p. 29–36.
36. Webster Jr, M., *Experimental methods*, in *Group processes*, M. Foschi and E.J. Lawler, Editors. 1994, Nelson-Hall Publishers. p. 43-69.
37. Shadish, W.R., T.D. Cook, and D.T. Campbell, *Experimental and quasi-experimental designs for generalized causal inference*. 2002: Houghton Mifflin.
38. Bengtsson, C., M. Persson, and P. Willenhag, *Gender and overconfidence.* Economics Letters, 2005. **86**(2): p. 199-203.
39. Puncochar, J.M. and P.W. Fox, *Confidence in individual and group decision making: When "two heads" are worse than one.* Journal of educational psychology, 2004. **96**(3): p. 582-591.
40. Moløkken-Østvold, K. and M. Jørgensen, *Group Processes in Software Effort Estimation.* Empirical Software Engineering, 2004. **9**(4): p. 315-334.
41. 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): p. 325-348.
42. Wilson, L.F. and N. Brekke, *Mental contamination and mental correction: unwanted influence on judgments and evaluation.* Psychological bulletin, 1994. **116**(1): p. 117-142.
43. Jørgensen, M. and K. Moløkken-Østvold. *Eliminating Over-Confidence in Software Development Effort Estimates*. in *Conference on Product Focused Software Process Improvement (PROFES)* 2004. Japan: Springer-Verlag. p. 174-184.
44. Johnson, D.D.P., *Overconfidence and war*. 2004: Harvard.
45. Steen, v.d.E., *Rational overoptimism (and other biases).* The American economic review, 2004. **94**(4): p. 1141-1151.
46. Manove, M., *Entrepeneurs, optimism and the competitive edge.* UFAE and IAE Working Papers 296.95, 1995.
47. Heine, S.J. and D.R. Lehman, *Cultural variation in unrealistic optimism: Does the West feel more invulnerable than the East?* Journal of personality and social psychology, 1995. **68**(4): p. 595-607.
48. Li, S. and Y. Fang, *Respondents in Asian Cultures (e.g., Chinese) are More Risk-Seeking and More Overconfident than Respondents in Other Cultures (e.g., in United States) but the Reciprocal Predictions are in Total Opposition: How and Why?* Journal of cognition and culture, 2004. **4**(2): p. 263-292.
49. Green, K.W.J., B. Medlin, and D. Whitten, *Developing optimism to improve performance: an approach for the manufacturing sector.* Industrial management & data system, 2004. **104**(2): p. 106-114.
50. Kruglanski, A.W., *Motivated social cognition: Principles of the interface*, in *Social psychology: Handbook of basic principles*, E.T. Higgins and A.W. Kurglanski, Editors. 1996, Guilford Press: New York. p. 493-520.
51. Shepperd, J.A., et al., *Exploring the causes of comparative optimism.* Psychologica Belgica, 2002. **42**(1/2): p. 65-98.
52. Buehler, R., D. Griffin, and H. MacDonald, *The role of motivated reasoning in optimistic time predictions.* Personality and Social Psychology Bulletin, 1997. **23**(3): p. 238-247.
53. Jørgensen, M. and K. Moløkken-Østvold, *Reasons for software effort estimation error: impact of respondent role, information collection approach, and data analysis method.* IEEE Transactions on Software Engineering, 2004. **30**(12): p. 993-1007.
54. Buehler, R. and D. Griffin, *Planning, personality, and prediction: The role of future focus in optimistic time predictions.* Organizational Behavior and Human Decision Processes, 2003. **92**: p. 80-90.
55. Buehler, R., D. Griffin, and M. Ross, *Exploring the "Planning fallacy": Why people underestimate their task completion times.* Journal of Personality and Social Psychology, 1994. **67**(3): p. 366-381.
56. Rozenblit, L. and F.C. Keil, *The misunderstood limits of folk science: an illusion of explanatory depth.* Cognitive Science, 2002. **26**(5): p. 521-562.