Relations between effort estimates, skill indicators, and measured programming skill

Magne Jørgensen[1,2]
Gunnar Rye Bergersen[3] and
Knut Liestøl[3]
[1]Simula Metropolitan Center for Digital Engineering, [2]Oslo Metropolitan University, [3]Department of informatics, University of Oslo

Abstract
There are large skill differences among software developers, and clients and managers will benefit from being able to identify those with better skill. This study examines the relations between low effort estimates, and other commonly used skill indicators, and measured programming skill. One hundred and four professional software developers were recruited. After skill-related information was collected, they were asked to estimate the effort for four larger and five smaller programming tasks. Finally, they completed a programming skill test. The lowest and most over-optimistic effort estimates for the larger tasks were given by those with the lowest programming skill, which is in accordance with the well-known Dunning-Kruger effect. For the smaller tasks, however, those with the lowest programming skill had the highest and most over-pessimistic estimates. The other programming skill indicators, such as length of experience, company assessed skill and self-assessed skill, were only moderately correlated with measured skill and not particularly useful in guiding developer skill identification. A practical implication is that for larger and more complex tasks, the use of low effort estimates and commonly used skill indicators as selection criteria leads to a substantial risk of selecting among the least skilled developers.

1. Introduction

«*I am wiser than this man, for neither of us appears to know anything great and good; but he fancies he knows something, although he knows nothing; whereas I, as I do not know anything, so I do not fancy I do. In this trifling particular, then, I appear to be wiser than he, because I do not fancy I know what I do not know.*» Socrates in Plato's *Apology* (399-389 BC)

Ideally, software developers would know their skill level and estimate the required work effort for carrying out tasks. This implies that clients and managers could safely select developers and evaluate their skill relative to other developers based on their effort estimates. This may occasionally be the case: A study of seven offshoring companies, with approximately equally good performance evaluations from previous clients and with similar experience level, demonstrated that the developers from the company with the lowest effort estimate spent the least effort and delivered the best quality [1]. The opposite, however, was observed in a study of four Norwegian companies, also developing the same software [2], where the developers from the company with the lowest effort estimate required the most effort and delivered the lowest quality. The finding that the least skilled developers can often be found among those with the lowest effort estimates is in accordance with the so-called Dunning–Kruger effect [3], a cognitive bias in which less skilled people also tend to be unaware of their lack of competence owing to poorer metacognitive skills, and therefore tend to be more over-optimistic in their predictions of their own performance. The original finding was on people's assessment of their skill relative to others' skill, but the effect has also been found to include other types of performance predictions [4].

The explanation of the Dunning–Kruger effect is still under debate. It has been proposed that it is caused by an anchoring-and-adjustment heuristic or a regression-towards-the-mean effect [5], and that it is a task-related artefact stemming from the more complex problem of task understanding by those with lower skill [6]. In spite of a lack of a commonly accepted explanation and the methodological problems of the original study [4], the Dunning–

Kruger effect itself, i.e. that lower skill is connected with more over-optimistic performance estimates, appears to persist across studies in academic [7], sport [8] and work-related [9] settings.

From prior studies, it is known that selecting software developers based on low effort estimates, or low-price bids derived from low effort estimates, increases the likelihood of selecting *over-optimistic* developers, which in turn increases the risk of project problems, as for example, in the study on 'the winner's curse' described in [10]. To the authors' knowledge, the relation between an emphasis on low effort estimates and the increased likelihood of selecting developers with *low programming skill* has not previously been investigated. It may be the case that the observed problems connected with developer or provider selection based on low effort estimates or prices are not only that the plans become unrealistic and the providers make financial losses and behave opportunistically [11], but also that a focus on low effort estimates leads to selecting less skilled software developers.

The above observations motivate the first research question:

RQ1*: What are the relations between effort estimates and programming skill in software development?*

Low-skilled software developers will not be selected if the skill assessments by the companies or the skill self-assessments by the developers, e.g. as described in their CVs, were reliable indicators of actual programming skills. This motivates the second research question:

RQ2*: How well do commonly used skill indicators, including company and self-assessed skill indicators, correspond with measured programming skill?*

The remainder of this paper describes the study design and characteristics of the collected data (Section 2), the results (Section 3), discusses the results, possible explanations of the findings and limitations of the work (Section 4) and concludes (Section 5).

2. Study design

2.1 Participants

Two software companies were contacted and asked to provide Java developers as participants in the study. One of the companies, located in Poland, had more than 500 software developers, had completed software projects for clients around the world for more than ten years. Their work processes were mainly based on agile development. The other company, located in Ukraine, was a branch of a larger international software development company with more than 1000 software developers. This company had mainly clients from US and Europe in sectors such as government, banking, real estate and tourism. The development processes were typically agile for product development.

A mixture of junior, intermediate and senior Java developers were requested. Junior, intermediate and senior were skill categories used by the companies to, amongst others, determine the hourly payment rates. A senior developer was typically compensated 20% more than an intermediate, and an intermediate 20% more than a junior. The companies were compensated for the participation of their developers using their ordinary hourly payment rates. It was indicated that the estimation and programming work would require four to five hours, and at least half a year of experience in Java development was a requirement.

Both companies accepted the request and offered, in total, 104 (57 + 47) software developers as participants. Among them, 27% were categorised by the companies as junior, 43% as intermediate and 30% as senior developers. Seven out of the 104 developers were female.

The steps of the study were as follows:

1. Collection of information about the participants (described below)
2. Provision of the estimates of each of the nine tasks (see Section 2.2)
3. Completion of the five smaller tasks (see Section 2.3)

All steps were typically collected in one session. If there was a need for a longer break, this should be done between Step 2 and Step 3. The full questionnaire of the information collection and estimates will be sent upon request.

The collection of information about the participants requested information about length of experience, self-assessed programming and effort estimation skill. The self-assessed skill measures used a scale from 1 (novice) to 5 (expert). Table 1 describes characteristics of the developers, including a separation into their payment category (company assessed skill-category).

Table 1: Education, experience and self-assessed general skill

Characteristic		Mean (stddev)	Median	Minimum	Maximum
Experience as software professional (years)	Total	7.5 (7.4)	6.0	0.7	27.0
	Junior	3.0 (2.2)	2.4	0.7	8.6
	Intermediate	7.3 (5.2)	6.0	2.1	26.0
	Senior	11.9 (5.4)	11.3	2.0	27.0
Experience as Java programmer (years)	Total	5.7 (4.4)	5.0	0.5	18.3
	Junior	2.0 (1.1)	1.6	0.5	5.0
	Intermediate	5.3 (2.9)	5.0	0.5	18.0
	Senior	9.7 (4.7)	10.0	1.0	18.3
Number of projects completed	Total	9.3 (7.8)	8	1	50
	Junior	6.9 (8.3)	4.5	1	40
	Intermediate	8.2 (4.4)	8.0	2	20
	Senior	13.1 (9.9)	10	4	50
Self-assessed general programming skill (1 … 5)	Total	3.5 (0.8)	4	2	5
	Junior	2.9 (0.8)	3	2	5
	Intermediate	3.6 (0.6)	4	3	5
	Senior	4.0 (0.7)	4	2	5
Self-assessed skill in Java-programming (1 … 5)	Total	3.7 (0.8)	4	2	5
	Junior	3.0 (0.9)	3	2	5
	Intermediate	3.9 (0.7)	4	2	5
	Senior	4.2 (0.7)	4	3	5
Self-assessed skill in estimation of software development effort (1 … 5)	Total	3.0 (0.9)	3	1	5
	Junior	2.3 (0.9)	2.5	1	4
	Intermediate	3.1 (0.7)	3	2	4
	Senior	3.4 (1.0)	4	2	5

Seventy percent of the developers had previously been responsible for estimating software projects, and 96% had been responsible for the estimation of their own work. As can be seen from Table 1, an increase in company assessed skill-category, e.g., from junior to intermediate or from intermediate to senior, corresponded with an increase in the mean and median length of experience, number of projects completed and self-assessed skill.

2.2 Task estimation

Following the provision of information about themselves, the software professionals were asked to estimate the effort, in work-hours and minutes, they thought it would most likely be required to complete nine different tasks using the programming language Java in their ordinary development environment. Four of the tasks were larger tasks (or smaller projects), where the participants were asked to develop new software systems using Java. The remaining five, which were later used in the evaluation of their programming skill, were smaller Java programming tasks in which, typically, existing Java code should be modified. The sequence of the estimation work was randomised for each participant, using the randomiser function in the survey tool Qualtrics (www.qualtrics.com). The same survey tool was used for the data collection of the estimates as well as the self-reported experience and skill information.

Table 2 presents the descriptions and estimates of the four larger tasks (Tasks A–D), including the median estimates per company-assessed skill category (junior, intermediate and senior). These tasks were only estimated and not completed. The table includes the actual effort by other developers to complete the tasks, when this information was available (Tasks A

and B). When the actual effort spent by other developers was not available (Tasks C and D), the median and interquartile ranges of estimates, as provided by developers *not* participating in this study, are presented.

The effort needed for the completion of the Task A-D will naturally vary from develop to developer, depending on, amongst others, tool support and relevance of competence. The median estimates in this study is, nevertheless, not far off from the actual or estimated effort values in other studies. This supports the impression that the estimation work was taken seriously by the developers in this study.

There was no clear connection between the median estimate of a task and the company-assessed skill category. Notice that those categorized as junior developers, i.e., those assessed to be least competent by the companies, had the lowest median estimates for Task A, C and D and the second lowest for Task B. The task specifications will be sent to interested readers upon request.

Table 2: Characteristics of the larger tasks (projects)

Task	Description	Actual effort or effort estimates by other developers	Effort estimates of developers in this study
A	Development of a web system with an interface to an xml file containing information about scientific articles.	Six different companies spent 29, 84, 187, 242, 474 and 527 (median 215) work hours, see [1].	The median estimated effort was 62 work hours. Median estimates of junior, intermediate and senior developers were, respectively, 40, 50 and 80 work-hours.
B	Development of a database with information about research studies, with links to other databases and user interface.	Four different companies spent 85, 90, 108 and 155 (median 99) work hours, see [2].	The median estimated effort was 158 work hours. Median estimates of junior, intermediate and senior developers were, respectively, 140, 160 and 120 work-hours.
C	Development of a web-based system that graphically displays on a world map the connections between different countries (such as number of outsourced projects).	The median estimated effort, as estimated by 78 software professionals, was 40 work hours, see control group estimates for Project A in [12].	The median estimated effort was 40 work hours. Median estimates of junior, intermediate and senior developers were, respectively, 40, 40 and 60 work-hours.
D	Development of a standalone desktop system with rule-based support for selection of jogging shoes.	The median estimated effort, as estimated by 126 software professionals, was 112 work hours, see control group estimates for Project B in [12].	The median estimated effort was 56 work hours. Median estimates of junior, intermediate and senior developers were, respectively, 49, 68 and 80 work-hours.

The smaller tasks (Tasks E–I) were first estimated and then completed by the developers. Table 3 presents the descriptions, actual completion efforts, proportion completed correctly and effort estimates for these tasks. It also includes a separation into the company-assessed skill-categories junior, intermediate and senior.

As can be seen, there is a tendency towards overly high estimates of the effort required to complete the smaller tasks amongst those providing a correct solution[1]. Based on the tasks with a correct solution, the median overestimation was as much as 80 min. An over-estimation of relatively small tasks is a frequently observed phenomenon, dating back as early as 1868 in results reported by Vierordt [13], and summarised for the context of effort estimates in [14]. The overestimation may, to some extent, be because the smaller tasks appeared more complex than they were in reality. For example, the inclusion of the Java code that should be understood and updated made the task material more extensive and complex than the task completion itself, as most of the tasks were fairly simple. It may also be that estimates of the smaller tasks,

[1] It is not meaningful to calculate the estimation error for incomplete or incorrect solutions, and these are consequently excluded from the estimation error measurement. It should be noticed that this leads to a selection bias affecting the error measurement, i.e. only the best developers are included.

when estimated after a larger task, are affected by the previous task estimates, i.e., a type of anchoring or assimilation effect increasing the effort estimates of the larger tasks [15]. More on possible explanations for the observed estimation over-pessimism of the smaller tasks in Section 3. As opposed to the larger tasks, there is no clear connection between company assessed skill-category (junior, intermediate and senior) and median estimates. The junior developers did worse than the intermediate and the senior on the two most complex tasks (Tasks G and I, i.e., the tasks with the lowest proportion of correct solution), but had otherwise similar median actual effort and proportion of correct solution.

Table 3: Characteristics of the smaller tasks

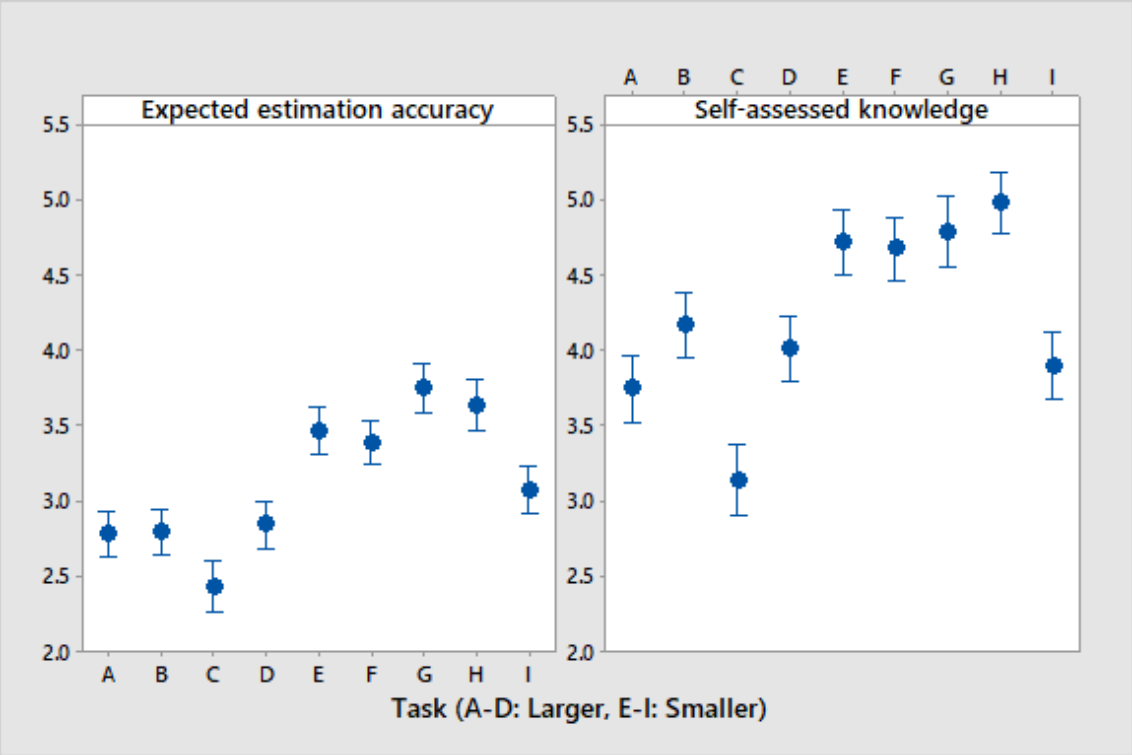
Task	Description	Median actual effort of the correct solutions and proportion of correct solutions	Estimated effort of the developers in this study
E	Adding functionality to an existing ATM application. Source code of the application was displayed.	The median actual effort was 25 min. Median estimates of junior, intermediate and senior developers were, respectively, 24, 25 and 21 minutes. Proportion of correct: 66%. Proportion correct for junior, intermediate and senior developers were, respectively, 70%, 63% and 67%.	The median estimated effort was 155 min. Median estimates of junior, intermediate and senior developers were, respectively, 180, 120 and 360 minutes.
F	Adding functionality to an existing program for a coffee vending machine. UML diagram and source code of the application was displayed.	The median actual effort was 8 min. Median estimates of junior, intermediate and senior developers were, respectively, 11, 12 and 11 minutes. Proportion of correct: 96%. Proportion correct for junior, intermediate and senior developers were, respectively, 100%, 93% and 95%.	The median estimated effort was 120 min. Median estimates of junior, intermediate and senior developers were, respectively, 120, 120 and 180 minutes.
G	Modifying a laser controller program. Source code of the application was displayed.	The median actual effort was 9 min. Median estimates of junior, intermediate and senior developers were, respectively, 13, 12 and 9 minutes. Proportion of correct: 43%. Proportion correct for junior, intermediate and senior developers were, respectively, 22%, 53% and 47%.	The median estimated effort was 60 min. Median estimates of junior, intermediate and senior developers were, respectively, 60, 30 and 60 minutes.
H	Writing unit tests for a program that listed the directory content of a specified root directory.	The median actual effort was 14 min. Median estimates of junior, intermediate and senior developers were, respectively, 21, 14 and 13 minutes. Proportion of correct: 74%. Proportion correct for junior, intermediate and senior developers were, respectively, 76%, 83% and 70%.	The median estimated effort was 90 min. Median estimates of junior, intermediate and senior developers were, respectively, 123, 60 and 120 minutes.
I	Fixing a bug and extend the functionality of a lab-order system of a healthcare related software.	The median actual effort was 39 min. Median estimates of junior, intermediate and senior developers were, respectively, 35, 37 and 37 minutes. Proportion of correct: 15%. Proportion correct for junior, intermediate and senior developers were, respectively, 4%, 19% and 20%.	The median estimated effort was 390 min. Median estimates of junior, intermediate and senior developers were, respectively, 600, 255 and 435 minutes.

Note: Task E–I contained 7, 12, 4, 37, and 4 Java files and consisted of 280 (46), 310 (59), 50 (61), 979 (74), and 104 (27) Lines of code (lines of comments) including unit test cases (Task H, I, ***) respectively. Tasks E–G had a "happy path" (https://en.wikipedia.org/wiki/Happy_path) output scenario in the task description. Task F had a UML sequence diagram. Tasks E–F had a PDF with the available source code for reference during the estimation task.

Following each task estimate, the developers provided the expected estimation accuracy using a scale from 1 (very low) to 5 (very high), and assessed their problem-solving knowledge using a scale from 1 (no idea of what to do) to 6 (know exactly what to do). Figure 1 shows the mean responses from these two assessments, with 95% confidence intervals of the mean

values. As can be seen, the estimates for the larger tasks (Tasks A–D) had typically lower expected accuracy compared with those for the smaller tasks (Tasks E–I); furthermore, the developers were less confident about their knowledge of how to solve the task. It can also be seen that the last (and most complex) of the smaller tasks (Task I) was estimated with less expected accuracy compared with the other smaller tasks; furthermore, the developers indicated lower problem-solving knowledge. The correlation between expected estimation accuracy and self-assessed knowledge was relatively strong (r = 0.66).

Figure 1. Confidence intervals of expected estimation accuracy and self-assessed knowledge per task



2.3 Measurement of programming skill

The tasks E-I were those solved by the developers for the purpose of measurement of programming skill. The tasks were solved in the same sequence by all developers starting with Task E and finishing with Task I. We used an online tool from Technebies (www.technebies.com) to support the skill measurement. The tool had previously been validated in terms of measurement and external validity [16]. Its compatibility with contemporary cognitive skill and ability theory has been verified as well [17].

Using the tool, each developer's performance is measured using the polytomous Rasch model [18], as implemented in the R package eRm (CRAN.R-project.org/package=eRm, see [19]). Briefly, the Rasch model estimates skill (ability) and task difficulty on the same, interval scale. Zero on this scale is by convention used as the mean difficulty of the available tasks. The mean difficulty can be compared with the mean skill of the subjects to detect whether the tasks as a whole were easy (skill > 0) or difficult (skill < 0) for the subjects. The unit of the scale is the logarithm of odds (logits[2]) and when skill equals difficulty, the probability of a correct answer to a task is 50%. Moreover, the scale can express differences in skill as odds. For example, a (relative) difference of, say, three logits in skill between two individuals would yield exp(3)[3]. This corresponds to approximately 20 to 1 in fair odds of the less skilled individual doing "better" on a programming task than the more skilled. For tasks H and I, which have not previously been reported in the literature, the main validation steps reported in

[2] logit(p) = log(p/1-p), where p is the probability and p/(1-p) is the odds.

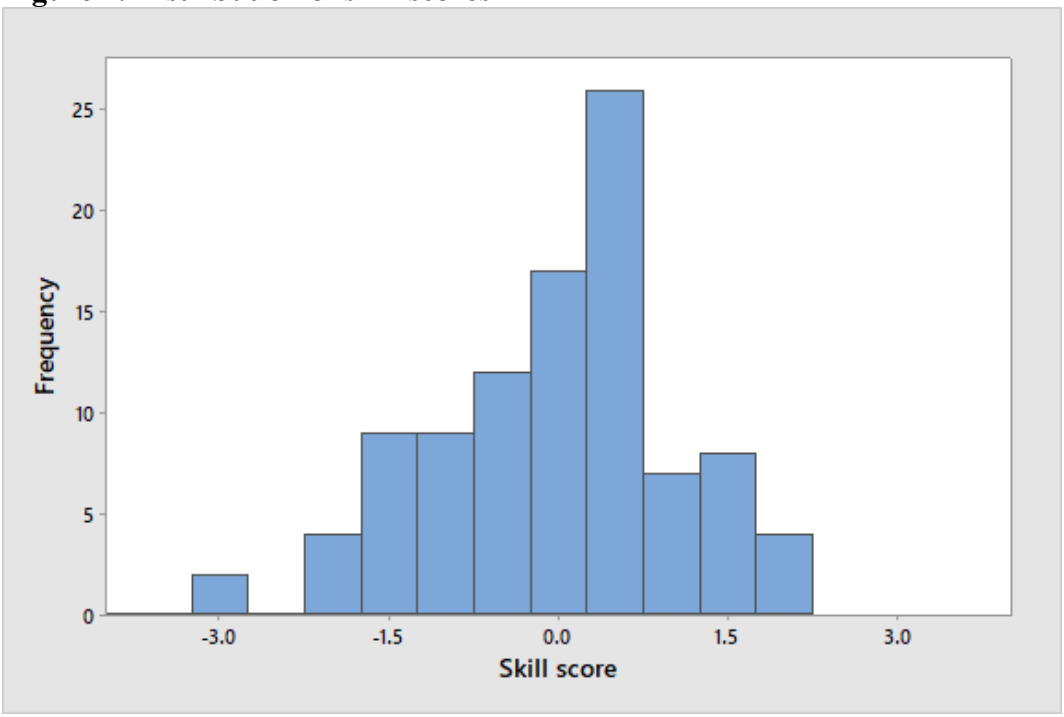
[3] $\text{logit}_1 = 3*\text{logit}_2 \Rightarrow \log(\text{odds}_1) = 3*\log(\text{odds}_2) \Rightarrow \text{odds}_1 = \exp(3)+\text{odds}_2$, which means that the difference in odds is exp(3)=20.09.

[16] were used for calibration and to ensure that programming skill was measured, as in the previously calibrated tasks E–G.

The tasks were completed using a regular integrated development environment (IntelliJ or Eclipse) on a remote Amazon Web Services (AWS) instance using virtualisation that allowed the developers to access their development environment through a browser. To prevent researcher bias, an engineer at Technebies automatically generated all skill measurements without any access to the developers' personal data, background information or effort estimates, and provided these results to the first author before the analysis reported in this paper was conducted.

The median skill score was −0.12, with an interquartile range of 1.50 = 0.62 − (−0.88). The quartiles of the skill score (−0.88, −0.12 and 0.62) were used to divide the developers into four skill categories (Q1–Q4) for later analysis. Figure 2 shows the distribution of the skill scores, excluding two developers whose programming performance was insufficient for proper calculation of skill scores. These two developers had no discernible progress on any of the five programming tasks and thus were put in the lowest skill category (Q1).

Figure 2. Distribution of skill scores



2.4 Analysis method

A linear mixed model [20] was used to analyse the relation between measured programming skill and effort estimates (RQ1). The variance estimation is based on restricted maximum likelihood, and the tests of fixed effects used the Kenward–Roger degrees of freedom approximation. The use of a mixed effect model enables repeated measurement and inclusion of the effect of both random and fixed variables. In the present model, Developer is considered a random variable; whereas, Task and Skill category are fixed variables. The model includes an interaction effect to examine whether there are task differences in how skill category affects the estimates. The variables are described in Table 4.

Table 4: Variables included in the analysis related to RQ1

Variable	Type	Description
lnEst = ln(Estimated effort)	Response variable	The estimated effort of a project. The estimates were strongly right-skewed with a few overly high values. Therefore, the natural logarithm of the estimate was used to ensure a more symmetric and closer to normal distribution of the estimates.

Developer	Random effect	Developer identification number (id1–id104). Each developer estimated all nine projects.
Task	Fixed effect	Task identification (A–E for the smaller, and F–I for the larger tasks).
Skill category	Fixed effect	The skill score was divided into skill categories Q1 (lowest skill) to Q4 (highest skill) using the skill quartiles as category limits. A division into four skill categories follows the initial and several follow-up analyses of the Dunning–Kruger effect [3].
Task x Skill category	Fixed effect (interaction variable)	The interaction between project and skill score category, examining whether there are different skill effects for different tasks.

The relation between skill and estimates may be different for the larger tasks A–D, where the estimates are based a requirement specification, and there appears to be a tendency towards underestimation, and the much smaller tasks E–I, where the estimates are typically based a short specification and pieces of Java-code, and there appears to be a tendency towards overestimation. In addition, it may be difficult to have a joint analysis of projects that would take days to complete (Tasks A–D) and smaller tasks requiring a few minutes (Tasks E–I). Accordingly, these two datasets were analysed separately, yielding 416 observations (104 developers × 4 tasks) for the first dataset (Tasks A–D), and 520 observations (104 developers × 5 tasks) for the second dataset (Tasks E–I).

An important concern of the study design is to avoid the regression effect biasing the results of the original Dunning–Kruger study [5]. The regression effect, caused by a mathematical coupling [21] between the independent and dependent variables, is a problem when programming skill is measured on the same tasks as those used to examine the relation between skill and task estimates, including degree of estimation bias. For the analysis of the data set with the four larger tasks, there is no such problem, as the skill score is derived from the developers' performance on the five smaller tasks combined. For the analysis of each of the five smaller tasks, however, this would include a slight mathematical coupling of the independent and dependent variables. Therefore, the programming performance score (used to measure skill) was excluded from the calculation of a developer's skill score on small tasks. This implies, for example, that the skill score of a software developer estimating Task F (one of the smaller tasks) is based on his/her performance on Tasks E, G, H and I (the remaining four smaller tasks), but not on Task F. Accordingly, the catR (CRAN.R-project.org/package=catR) package was used to measure skill based on the task difficulty parameters from eRm, whereas the programming performance observation of the task being estimated was treated as missing.

The analysis related to the second research question (RQ2) examines how strongly the following company- and self-evaluations of skill are connected with the measured programming skills:

- Skill/payment category (junior, intermediate, senior) of developer as assessed by the company.
- Length of experience as software developer in general.
- Length of experience as software developer using Java.
- Self-assessed programming skill in general.
- Self-assessed programming skill using Java.
- Self-assessed effort estimation skill.
- Confidence in accuracy of effort estimates (per task).
- Confidence in knowledge related to problem solving (per task).

Concordance-based measures were used, i.e. Somers' D and the ability to predict the more skilled developer out of two ('hit rate'), together with Spearman rho (rank-based correlation coefficient) for the analysis of the relation between a skill indicator and measured programming skill.

3. Results

3.1 Relation between skill category and estimates (RQ1)

Linear mixed models were used, with the estimate (lnEst) as the response variable, Developer as random variable, and Task, Skill category and the interaction between skill category and task as dummy-coded (0 or 1) fixed variables for the set of larger and for the set of smaller tasks. The model parameters and residuals, which were close to being normally distributed, are included in Appendix 1. The adjusted R^2-values were 78% for the larger and 74% for the smaller tasks.

Table 5 shows the results of the fixed effects test and the Tukey pairwise comparison of the mean estimate for each skill category. Box plots displaying the effort estimates according to skill category, aggregated for the larger and the smaller tasks, are shown in Figures 3 and 4. Figures 5 and 6 show the connection between the skill scores and lnEst, per task, using the LOWESS (locally weighted smoothing) smoother function with 0.5 degrees of smoothing and two iterations (steps). For presentation purposes, two outliers with low (less than −3) programming skill are not displayed.

Table 5: Test of fixed effects and Tukey pairwise comparison of skill categories

<table>
<tr><th>Item</th><th colspan="2">Larger tasks</th><th colspan="2">Smaller tasks</th></tr>
<tr><td>Task</td><td colspan="2">F-value 25.3, p-value < 0.001</td><td colspan="2">F-value 68.8, p-value < 0.001</td></tr>
<tr><td>Skill Category</td><td colspan="2">F-value 2.92, p-value = 0.038</td><td colspan="2">F-value 8.33, p-value < 0.001</td></tr>
<tr><td>Task x Skill Category</td><td colspan="2">F-value 1.75, p-value 0.08</td><td colspan="2">F-value 1.94, p-value 0.028</td></tr>
<tr><td>Tukey pairwise comparisons of mean estimates (back-transformed from lnEst) of the four skill categories.</td><td colspan="2">Q1: 43 h
Q2: 95 h
Q3: 107 h
Q4: 66 h</td><td colspan="2">Q1: 278 min
Q2: 143 min
Q3: 128 min
Q4: 112 min</td></tr>
<tr><td rowspan="5">Tukey pairwise comparison of mean estimates (back-transformed from lnEst) for Task x Skill Category, together with the reference effort.

The reference effort for Task A and B is based on the median actual effort of other companies solving the task, whereas that of Tasks C and D on the median estimated effort by other developers.

The reference efforts for Tasks E–I (Actual) are the median actual efforts of those completing the tasks with a correct solution. The proportion of correct solutions per task and skill category is shown in round brackets. As before (relevant for the smaller tasks only), the skill score excludes the score on the task being estimated.</td><td>Task A</td><td>Q1: 40 wh
Q2: 64 wh
Q3: 98 wh
Q4: 58 wh
Reference effort: 215 wh</td><td>Task E</td><td>Q1: 380 min
Actual: 23 min (42%)
Q2: 213 min
Actual: 22 min (50%)
Q3: 161 min
Actual: 27 min (73%)
Q4: 118 min
Actual: 19 min (83%)</td></tr>
<tr><td>Task B</td><td>Q1: 61 wh
Q2: 225 wh
Q3: 177 wh
Q4: 129 wh
Reference effort: 155 wh</td><td>Task F</td><td>Q1: 293 min
Actual: 16 min (83%)
Q2: 105 min
Actual: 11 min (100%)
Q3: 151 min
Actual: 11 min (96%)
Q4: 98 min
Actual: 10 min (100%)</td></tr>
<tr><td>Task C</td><td>Q1: 45 wh
Q2: 72 wh
Q3: 84 wh
Q4: 46 wh
Reference effort: 40 wh</td><td>Task G</td><td>Q1: 152 min
Actual: 14 min (11%)
Q2: 55 min
Actual: 12 min (42%)
Q3: 45 min
Actual: 10 min (54%)
Q4: 60 min
Actual: 9 min (59%)</td></tr>
<tr><td>Task D</td><td>Q1: 32 wh
Q2: 78 wh
Q3: 92 wh
Q4: 55 wh
Reference effort: 112 wh</td><td>Task H</td><td>Q1: 191 min
Actual: 15 min (43%)
Q2: 93 min
Actual: 10 min (93%)
Q3: 87 min
Actual: 10 min (81%)
Q4: 66 min
Actual: 9 min (95%)</td></tr>
<tr><td colspan="2"></td><td>Task I</td><td>Q1: 511 min
Actual: 36 min (7%)</td></tr>
</table>

			Q2: 455 min Actual: 39 min (16%) Q3: 361 min Actual: 39 min (14%) Q4: 392 min Actual: 32 min (24%)

Figure 3. Boxplot of estimates per skill category for the larger tasks

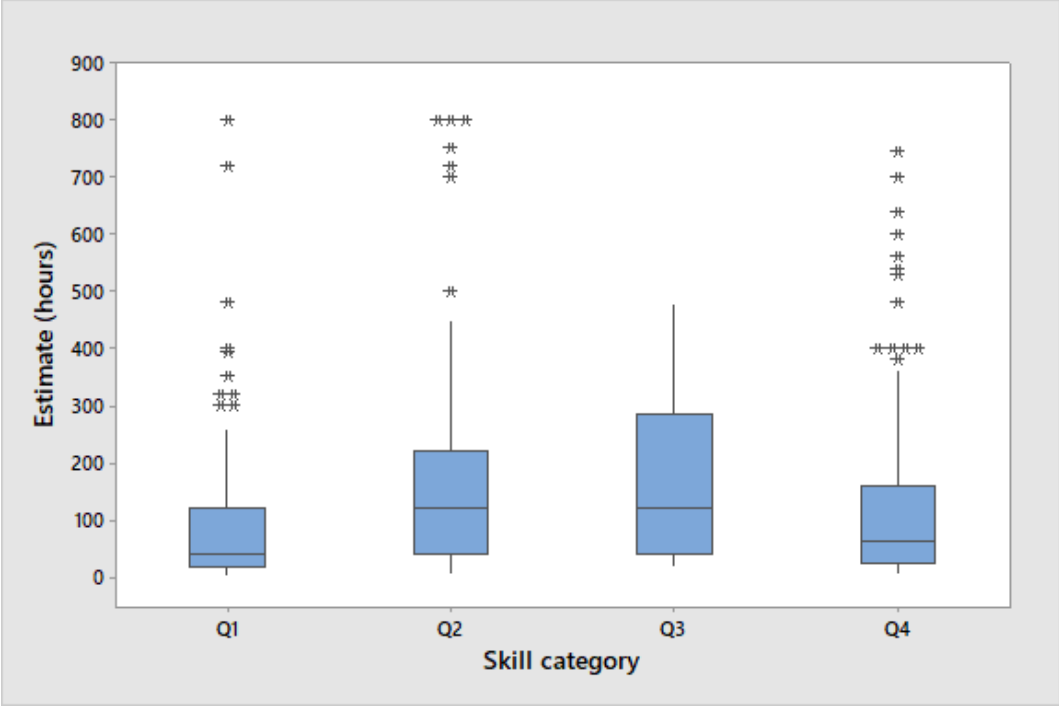


Figure 4. Boxplot of estimates per skill category for the smaller tasks

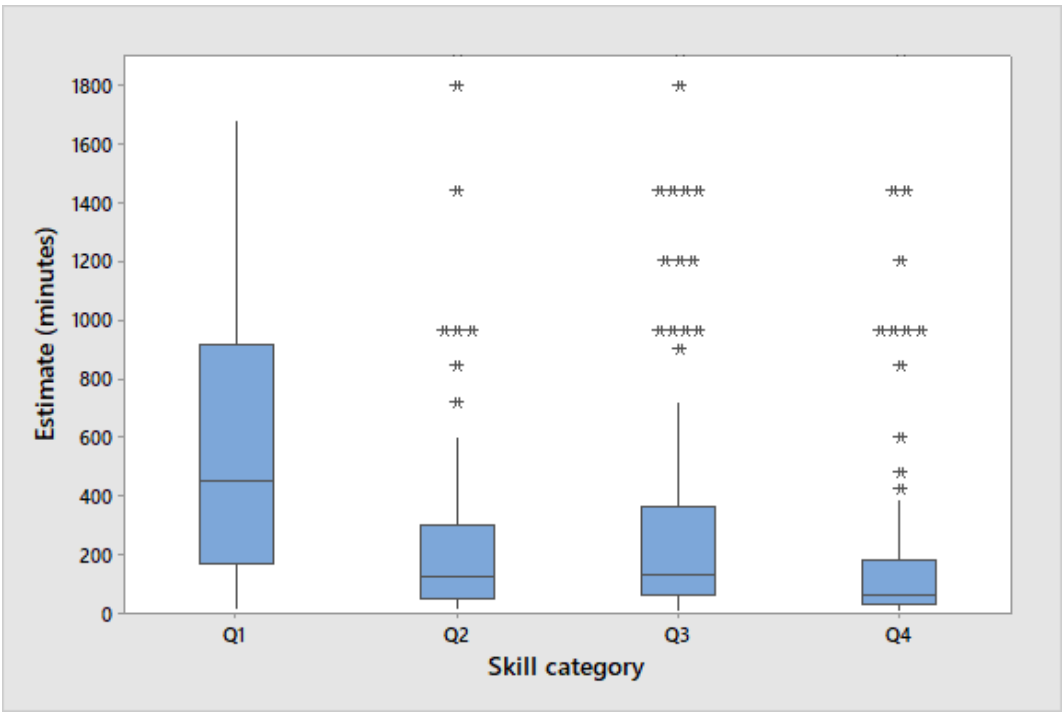


Figure 5. LnEst vs. skill score for the larger tasks

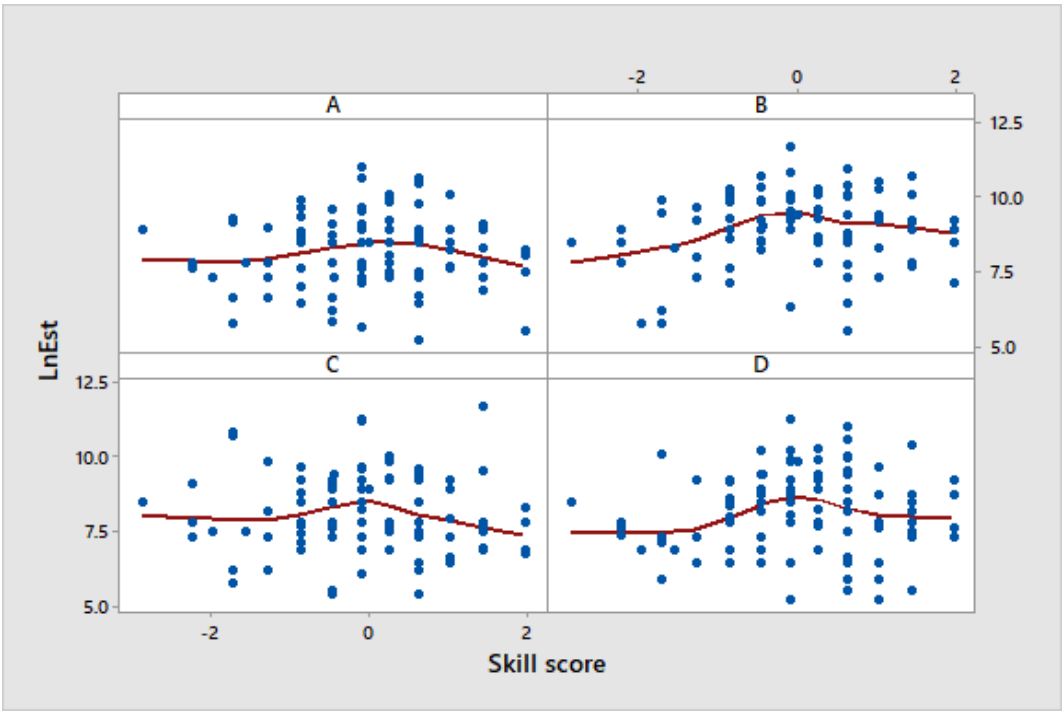
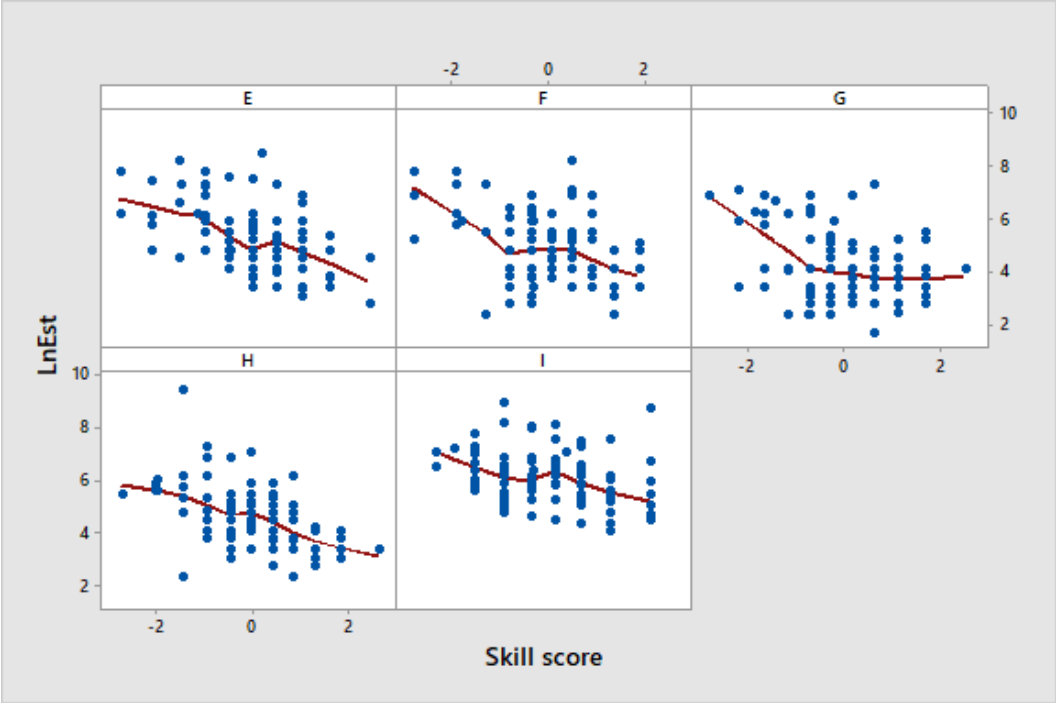


Figure 6. LnEst vs. skill score for the smaller tasks



For the four larger tasks (Tasks A−D, second column of Table 5 and Figures 3 and 5), the developers with the lowest skill scores frequently had the lowest estimates. In fact, those in the lowest skill category (Q1) had, on average, estimates lower than those in the highest skill category (Q4). Comparing the median estimates of the least skilled developers with the reference efforts, i.e. the effort used by other developers, suggests that the estimates of the least skilled tended to be strongly over-optimistic. This finding is in accordance with what is typically reported in studies examining the Dunning−Kruger effect, i.e. that lower skill is connected with stronger over-optimism regarding one's own abilities.

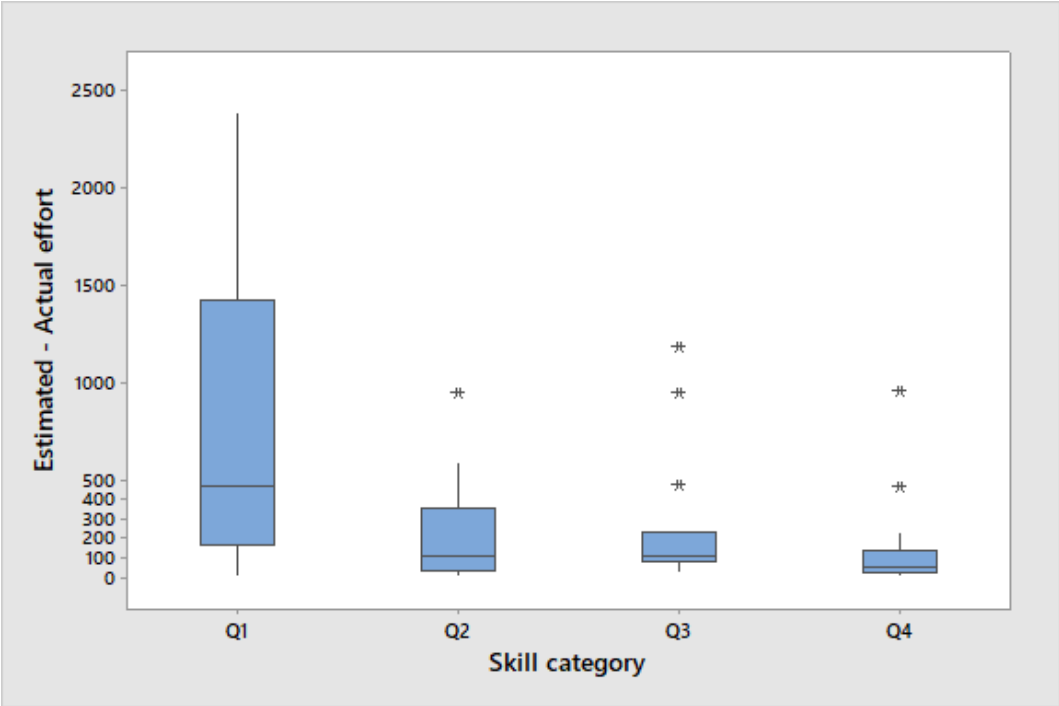
The results for the smaller tasks (Tasks E−I, third column in Table 5 and Figures 4 and 6) differ from those for the larger ones. For the smaller tasks, the developers with the lowest skill scores (Q1) had, on average, significantly higher effort estimates than those with better programming skill scores (Q2−Q4). This finding is perhaps best illustrated in Figure 6, which shows a decrease in lnEst as skill score increases for all five tasks. Comparing the effort estimates for the smaller tasks with the actual effort required to complete them, as presented in Table 5, it is inferred that all four skill categories have a tendency towards strong over-estimation, and that this over-pessimism is stronger for those with low programming skill.

The analysis of the relation between the degree of overestimation and skill score is, as previously noted, complicated by the fact that the developers with the lowest skill score had a lower proportion of correct solutions, and that the effort that would be required to correctly complete the tasks by those without a correct solution is unknown. Considering, for example, Task H, it is seen that 43% of those in skill category Q1 and 95% of those in skill category Q4 handed in a correct solution. This implies that the overestimation of the best performing 43% developers in the lowest skill score category is compared with that of the best performing 95% developers in the highest skill category. For the more complex smaller tasks, i.e. those with low proportion of correct solutions, a conclusion regarding the relation between skill score and over-pessimistic effort estimates should be carefully drawn. For the smaller tasks with higher completion rate, in particular Task F, the interpretation is more robust, suggesting that those in the lowest skill category indeed made more over-pessimistic effort estimates.

Figure 7 shows the relation between estimates and actual effort (in minutes) per skill category for Task F, which had a correctness rate of 96%, i.e., nearly all developers handed in a correct solution for this task. This task requested the development of added functionality of a coffee vending machine, and displayed 310 lines of code and a UML sequence diagram, i.e., there was much text and other information available to process when estimating the effort. This led nearly all developers to, as displayed in Figure 7, to over-estimate the effort. The degree of

over-estimation was, however, much larger in the lowest skill category. The median overestimation was 464 min for those in Q1, 104 min for those in Q3, 108 min for those in Q3 and 48 min for those in Q4.

Figure 7: Overestimation of effort per skill category (Task F)



A mechanism possibly explaining the much stronger over-pessimism within the lowest skill category for task F is that the lower programming skill also led to lower skill in identifying the level of difficulty of the relatively simple programming task. Those in the lowest skill category may have assessed the task to be medium complex or complex and failed to understand that the task was actually quite simple. This mechanism will be further discussed in Section 4.

3.2 Relation between self-reported skill indicators and measured programming skill (RQ2)

Table 6 shows results on how well company and self-reported programming skill indicators corresponded with the measured skill of the developers, i.e., their measured skill score based on completing the five programming tasks (Tasks E–I). The columns with skill categories show the proportion of developers in a category, or the median/mean values of the skill indicator for each of the programming skill quartiles (Q1–Q4). The median values are shown if there are a few high observations leading to a strongly right-skewed distribution, i.e. for the indicators in rows b) and c); otherwise, the mean value is used.

The asymmetric Somers' D [22] is calculated for all possible pairs of developers, i.e. $n * (n - 1) / 2$ pairs, where n is the number of developers. Pairs with the same value of the skill indicator (predictor) variable are removed. The remaining are divided into pairs for which the developer with higher value of the predictor variable also has a higher value of the dependent variable (termed concordant pairs (C)), pairs for which the developer with higher value of the predictor variable has lower value of the dependent variable (termed discordant pairs (D)), and pairs for which the developers have the same value of the dependent variables (tied pairs (T_d)[4]). Somers' D is then defined as $(C - D) / (C + D + T_d)$, i.e., as the difference between the proportion of concordant and discordant pairs. To facilitate its interpretation in terms of how

[4] Nine percent of the developer pairs where such ties, i.e. had different skill indicator value, but the same value for the measured programming skill.

well a skill indicator predicts the measured programming skill, a measure of the proportion of correct predictions was added to the sum of correct and incorrect predictions, defined as C / (C + D). This measure, which we termed *hit rate*, is similar to the Goodman and Kruskal gamma statistics (G) [23] in that it exclude all ties, i.e., it implicitly assumes that ties on the dependent variable (T_d) are neither indicating correct (hit) nor incorrect predictions (non-hit) of the dependent variable. To be useful as an indicator variable, it should enable a hit rate substantially different from 50%, which is what a random choice would result in. The p-values are tests of concordance, i.e. the probability of making the same observation even if there were no difference between the proportion of concordant and discordant pairs.

The two right-most columns show the Spearman rho (rank-based correlation coefficient), which in the present case may be a better measure of correlation than the Pearson correlation coefficient, as ordinal scales are used for the skill indicators. The columns show the correlation coefficient for all data and the correlations within each of the two companies (C1 and C2) employing the participating developers. If skill is assessed (either by the company or by the developer himself/herself) relative to other developers within the same company, one would expect stronger within-company correlations compared with the correlations found when developers from both companies are included.

Table 6: Evaluation of programming skill indicators

Skill indicator		**Skill category Q1**	**Skill category Q2**	**Skill category Q3**	**Skill category Q4**	**Somers' D and hit rate**	**Correlation all data**	**Correlation within company**
a) Company-assessed skill category[i]	Junior Intermed. Senior	29% 13% 26%	29% 27% 6%	36% 31% 39%	7% 29% 29%	D = 0.13 hit = 57% (p = 0.08)	r = 0.14 (p = 0.16)	C1: r = 0.19 (p = 0.16) C2: r =0.12 (p = 0.43)
b) Length of experience as software developer (median number of years)		4 years	4 years	8 years	7 years	D = 0.18 hit = 60% (p = 0.004)	r = 0.28 (p = 0.005)	C1: r = 0.22 (p = 0.11) C2: r = 0.44 (p = 0.003)
c) Length of experience with programming in Java (median number of years)		3 years	2.5 years	5 years	6 years	D = 0.21 hit = 62% (p < 0.001)	r = 0.31 (p = 0.002)	C1: r = 0.27 (p = 0.05) C2: r = 0.43 (p = 0.004)
d) Self-assessed general programming skill (mean values, scale: 1 (low) ... 5 (high))		3.3	3.3	3.5	4.0	D = 0.27 hit = 65% (p = 0.002)	r = 0.29 (p = 0.003)	C1: r = 0.30 (p = 0.02) C2: r = 0.37 (p = 0.013)
e) Self-assessed programming skill in Java (mean values, scale: 1 (low) ... 5 (high))		3.4	3.4	3.8	4.2	D = 0.26 hit = 64% (p = 0.003)	r = 0.27 (p = 0.006)	C1: r = 0.34 (p = 0.01) C2: r = 0.30 (p = 0.05)

f) Self-assessed skill in effort estimation (mean values, scale: 1 (low) ... 5 (high))	3.1	2.8	2.9	3.3	D = 0.10 hit = 56% (p = 0.13)	r = 0.13 (p = 0.21)	C1: r = 0.34 (p = 0.01) C2: r = 0.03 (p = 0.83)
g) Confidence in accuracy of one's own effort estimates (mean of all nine tasks, scale 1 (low) ... 5 (high))	2.9	3.2	3.1	3.3	D = 0.21 hit = 61% (p = 0.002)	r = 0.29 (p = 0.004)	C1: r = 0.27 (p = 0.04) C2: r = −0.06 (p = 0.70)
h) Confidence in problem-solving knowledge (mean of all nine tasks, scale 1 (no idea of what to do) ... 6 (know exactly what to do))	3.8	4.2	4.2	4.7	D = 0.26 hit = 64% (p < 0.001)	r = 0.37 (p < 0.001)	C1: r = 0.45 (p < 0.001) C2: r = 0.36 (p = 0.019)

i) For Somers' D, hit rate and correlational analysis, the company-assessed skill categories were assumed to be ordinal, i.e. junior < intermediate < senior.

The company-assessed programming skill level (Table 6, row a), as reflected in the junior, intermediate and senior programmer skill categories, was a weak indicator of measured programming skill, with a hit rate of 57% and correlation coefficient of 0.14. The relatively high proportion (26%) of senior developers in the lowest programming skill category (Q1) contributes to this. The correlation improved slightly in the within-company analysis for the company C1 (r = 0.19), but not for C2 (r = 0.12). A comparison of the employee skill levels of the two companies demonstrated that the median skill scores of junior, intermediate and senior programmers from C1 were −0.02, 0.62 and 0.62, respectively, whereas the corresponding scores for C2 were −0.88, −0.30 and 0.12. That is, an average junior developer in C1 had higher programming skill than an intermediate and close to that of a senior developer in C2. This suggests that if the average employee skill level for a company is known, the company-assessed competence level is even less useful as an indicator of the actual programming skill. One may argue that by the classification into junior, intermediate and senior programmers, the companies did not intend to indicate programming skill, but rather to represent experience level (seniority) or ability to work without support from more senior staff. Even if this should be the case from the company viewpoint, it is conceivable that several clients may interpret the junior–intermediate–senior categorisation as indicating skill, i.e., as resulting in better productivity and/or quality, as they pay more for intermediate and senior than for junior developers.

The length of programming experience in general (Table 6, row b), the length of programming experience with Java (Table 6, row c), the self-assessed programming skill in general (Table 6, row d) and the self-assessed programming skill in Java (Table 6, row e) had hit rates between 60% and 65% and correlation coefficients between 0.27 and 0.31, i.e. they were better than company-assessed skill levels, but not particularly strong indicators of programming skill, either. For one of the companies (C2), the length of programming experience in general as well as in Java were moderately good skill indicators, with correlation coefficients of 0.44 and 0.43, respectively. This suggest that there may be within-company contexts in which the correlation between length of experience and skill justifies using the former as a skill indicator. The challenge is, of course, to know when this is the case and when not.

The self-assessed skill in effort estimation (see Table 6, row f) was a weak indicator of programming skill, with a hit rate of 56% and correlation coefficient of 0.13. Interestingly, those in the lowest (Q1) programming skill category assessed themselves, on average, to have better estimation skills than those in the second lowest and second highest (Q2 and Q3) skill categories. In Section 3.1, it was pointed out that those with the lowest programming skill (Q1) were likely to have the least accurate effort estimates both for large and small tasks. This suggests that there is a Dunning–Kruger effect related to effort estimation skill, i.e. those with the lowest skill in effort estimation tend to over-estimate their estimation skill the most.

The mean confidence in the accuracy of the estimates was not particularly high for any of the skill categories (see Table 6, row f). Using confidence in the accuracy of the estimates as a skill indicator yielded a hit rate of 61% and a correlation coefficient of 0.29. Limiting the analysis to the estimation accuracy confidence for the smaller tasks, i.e. to the tasks used to measure programming skill, the relation between estimation accuracy confidence and programming skill weakened further, with a hit rate of 54% and correlation of only 0.10. The within-company correlations were lower than those across the total population. One of the companies (C2) even had a slightly negative correlation (−0.06) between confidence in the accuracy of the effort estimate and the measured programming skill.

The mean confidence in problem-solving knowledge (Table 8, row h) had the highest correlation with the measured programming skill (0.37) and the highest hit rate (64%) among the skill indicators, when the full data set was included. The relation was slightly stronger when the mean confidence in problem-solving knowledge was examined for the smaller tasks only, with a hit rate of 66% and a correlation of 0.39. The within-company correlation coefficients were approximately the same (0.36 for C2) or improved (0.45 for C1).

As previously observed, the lowest effort estimates typically belonged to those with the lowest skill for larger tasks and to those with the highest skill for smaller tasks. This suggests that the use of effort estimates as a programming skill indicator is not generally reliable, i.e. the relation may be negative in certain context, and positive in others. Nevertheless, there may be individual tasks in which the relation between effort estimate and programming skill is strong. The analysis of Somers' D, the hit rate and the Spearman rho rank-based correlation coefficient (Table 7) examines this.[5]

Table 7. Use of the estimate as indicator of programming skill per task

Task	Somers' D and hit rate	Correlation between estimate and skill
Task A	D = −0.05 hit = 47% (p = 0.22)	r = 0.08 (p = 0.42)
Task B	D = −0.11 hit = 44% (p = 0.06)	r = 0.15 (p = 0.13)
Task C	D = 0.01 hit = 50% (p = 0.53)	r = −0.01 (p = 0.94)
Task D	D = −0.09 hit = 45% (p = 0.08)	r = 0.13 (p = 0.19)
Task E	D = 0.34 hit = 68% (p < 0.01)	r = −0.41 (p < 0.01)
Task F	D = 0.26 hit = 64% (p < 0.01)	r = −0.35 (p<0.01)
Task G	D = 0.24 hit = 63%	r = −0.29 (p < 0.01)

[5] As before, to avoid regression effects, we base the skill measurement on the performance of all tasks except the one being estimated for all the analyses.

	($p < 0.01$)	
Task H	D = 0.37 hit = 70% ($p < 0.01$)	$r = -0.50$ ($p < 0.01$)
Task I	D = 0.22 hit = 62% ($p < 0.01$)	$r = -0.29$ ($p < 0.01$)

The results in Table 7 confirm the previous observation that developers with lower skill underestimated the larger tasks and overestimated the smaller tasks, compared with those with higher skill. More interestingly, the use of higher estimates as an indicator of lower programming skill yielded relatively high hit rates (62%–70%) for the smaller tasks. The highest hit rate and correlation was achieved for Task H, with a hit rate of 70% and correlation of −0.50. Comparing the proportion of correct solutions for the smaller tasks, Tasks H was neither among the most complex nor among the simplest. Task H appeared, however, to be sufficiently complex to separate developers with higher skill, who better understood that the task would require relatively low effort to complete, and developers with lower skill, who were unable to realise that.

A possible practical implication of this observation is that it may be possible to use effort estimates for tasks similar to Task H (i.e. tasks for which the skilled, but not the unskilled understand that no significant effort is required) as a fairly good skill indicator. That is, the effort estimates for seemingly complex tasks, in which higher skills are necessary for understanding their simplicity, may be the best indicator of programming skill, rather than the effort estimates for truly complex or obviously simple tasks. This is a simpler and less costly method than requesting software developers to complete programming tasks to assess their skills. There may, however, be threats to the use of estimates as skill indicators, e.g., if the developers strategically change their estimates to get a better skill score. How to design skill tests based on estimates may nevertheless, be an interesting topic for further research.

4. Discussion and limitations

4.1 Similarities and differences with findings in related studies

There are numerous prior studies comparing self-assessed skill with more objective skill criteria, such as actual performance on relevant tasks, e.g. [24]. To the authors' knowledge, none of these studies has examined the relation between self-assessed skill in the format of work effort estimates and measured work completion skill, or how well self-assessed programming skill is connected with measured programming skill.

Starting with the obvious, studies tend to find that people differ in actual skill. In software development, they usually differ substantially. The study reported in [25], where the bug fixing productivity of more than 200 programmers with a large software provider over 12 years is analysed, found that the 27% most productive programmers did 78% of the work, and the single most productive programmer, in spite of receiving the most complex bugs to fix, completed as much as 8.3% of the bug fixes. In the study reported in [1], a difference of 1:18 in actual effort spent for the development of the same software was found among seven providers. The level of variance in actual performance on the programming tasks in the present study is consequently not surprising. Similarly, the large variance in the effort estimates for the same tasks is consistent with the findings of prior studies [1, 12].

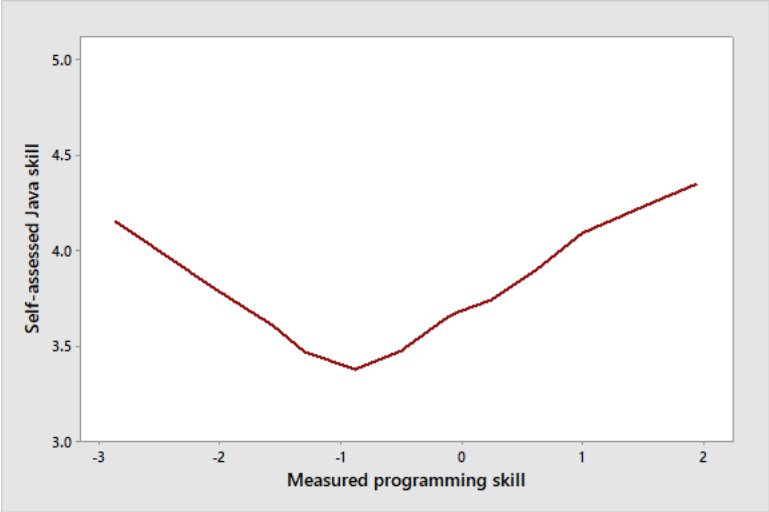
The finding that people generally tend to overestimate their skill has been reported in several studies. For instance in [26], it was demonstrated that in various domains, people tend to overestimate their skills, underestimate their shortcomings and do not let negative feedback affect their self-evaluation. The same tendency is also documented for effort estimation of software development projects [14, 27, 28], although not necessarily for smaller programming tasks, as pointed out in [14]. The findings related to the larger tasks, which indicate that people tend to underestimate effort and implicitly overestimate problem-solving ability, are consistent with the majority of prior software development and other studies. Kruger and Dunning [3] extended the above findings by observing a stronger level of over-optimism regarding own

skill among those with lower skill. They argue that '*incompetence … not only causes poor performance but also the inability to recognize that one's performance is poor*' (p. 1130). They also document that an increase in competence reduces the over-optimism of the poorest performers.

A contribution of the present study is to add software development effort estimation to the domains where the Dunning–Kruger effect of more over-optimism among those with lower skill has been documented. However, we observed the Dunning–Kruger effect only for the estimation of the larger tasks. For the smaller tasks, those with the lowest programming skill appear to have been more over-pessimistic about their own problem-solving ability. It is argued that the results may be seen as an extension of the finding of Kruger and Dunning, i.e., that there are situations where low skill leads to more over-pessimism rather than over-optimism. Possible reasons for the estimation difference between the larger and the smaller tasks are discussed in Section 4.3. It should be noted that both results support the basic claim by Kruger and Dunning, i.e., that lower skill is connected with lower ability to assess one's own skill level. The extension is mainly related to that lower ability to assess one's own skill level in relation to a task sometimes leads to over-pessimism in how much effort one would need to solve a task.

The robustness of the finding that low measured skill is connected with higher perceived self-assessed skill, i.e., the main claim by Kruger and Dunning, is illustrated in Figure 8. Figure 8 shows the relation between measured programming skill and self-assessed Java programming skill using a LOWESS smoother function (with a degree of smoothing set to 0.5 and two steps). As can be seen, the self-assessed Java programming skill increased with lower measured programming skill for those in the bottom half of skill scores (less than −1), but increased with higher measured programming skill for those in top half (more than −1).

Figure 8. Measured programming skill vs. self-assessed skill



The companies' skill categorisation of programmers as junior, intermediate or senior was found to be a poor indicator of measured programming skill (r = 0.14), even poorer than total length of experience as programmer (r = 0.28). The correlations for the same indicators reported in [16] were similarly poor, but here developer category had a stronger correlation to skill (r = 0.32) than total length of programmer experience (r = 0.15). A difference in how well programming skill can be predicted from the company-assessed skill category between contexts may not be surprising, given that different companies have different strategies for moving programmers from one skill and payment category to another. A difference in the correlation of the total length of experience with actual programming skill from one context to another is not surprising either. We observed, for example, that one of the companies (C2) had a significantly higher correlation between total length of experience and skill than the other another (r = 0.44 vs. r = 0.22). Both studies demonstrate that neither company-assessed skill category nor length of experience are strong indicators of measured programming skill. For instance, the correlation of 0.28 between total length of programming experience and

programming skill corresponds to a hit rate of only 60%, i.e. in four out of ten cases, between two programmers, the one with the longest experience is the least skilled.

The remaining skill indicators, i.e. self-assessed programming and estimation skill, confidence in the accuracy of the estimates and confidence in problem-solving knowledge, had hit rates in the range of 56%–64% and correlations in the range of 0.13–0.37. This is in accord with results reported for other domains. For example, in the meta-synthesis in [29], summarising the results from 22 meta-analyses covering more than 2000 studies on the correlation between self-evaluation and objective measures of actual performance, a mean correlation of 0.29 is reported. Consequently, correlations similar to those in the present study are quite expected in terms of the relations between self-assessed skill and measured performance. Unfortunately, this makes such skill indicators, which are widely used in software industry, not particularly useful in the identification and selection of the more skilled software developers.

4.2 Explanations

Dunning and Kruger [3] explained their observations by a relation between lower task completion skill and lower awareness of one's own problem-solving ability, i.e., lower skill is connected with lower skill-related meta-knowledge. As reported in Section 4.1, our data supports that the less skilled over-estimated their own skill level. The results on the larger programming tasks also support that lower skill may be connected with more over-optimistic effort estimates and more optimistic assessment of one's own estimation skills. However, it was observed that lower skill could also lead to more over-pessimistic performance estimates, i.e., the effort estimates of the smaller tasks seem to have been more over-pessimistic for those with lower skill. This calls for explanations extending those provided for the Dunning–Kruger effect. It is argued in the present study that lower skill, and the related lower meta-knowledge regarding one's own skill, do not necessarily lead to more over-optimism, but primarily to higher estimation uncertainty.

For truly complex tasks, a developer with low task completion skill may not identify and understand a task's *complexity*. However, for seemingly complex tasks, the same low task completion skill may lead to failure to identify and understand a task's *simplicity*. For example, several of the smaller programming tasks in the present study may have given an impression of being quite complex owing to the size and complexity of the code presented (up to 37 files and 979 lines of code), but were in reality fairly simple. Less-skilled developers were less likely to discover that the tasks did not require significant code changes, and therefore they strongly over-estimated the required effort. Identifying the actual simplicity level of the smaller tasks is indeed challenging, as even those with higher skill tended to overestimate effort, albeit to a lesser extent than those with lower skill.

Furthermore, previous studies suggest that when estimating task effort, software developers, consciously or unconsciously, recall actual effort on previous tasks, with an emphasis on tasks perceived to be similar to the task to be estimated [30, 31]. If a developer with low skill fails to identify the simplicity of a task, then most probably the experience from larger or more complex tasks is used as input and lead to an over-pessimistic estimate. As the task uncertainty increases, developers tend to make estimates closer to the average of the class of perceived relevant objects [32-34], a tendency termed "the central tendency of judgement". This may add to the over-pessimism of those with less skill in high-uncertainty situations, where the wrong class of relevant experience is chosen as reference for the effort estimates and the task looks more complex than it really is.

4.3 Implications for practice and further research

Previous studies document that emphasising low effort estimates, or the low pricing derived from such estimates in the selection of developers or providers increases the risk of selecting those with over-optimistic effort estimates [10, 35], thus increasing the risk of problematic software development [11, 36-38]. This study adds to those findings by documenting that selecting among developers with low effort estimates also increases the risk

of selecting less skilled developers, at least for larger tasks, which may be the majority in most relevant real-world contexts. If, for example, a client selected a developer among those with the 25% lowest effort estimates to complete Tasks A–D, the risk of making a selection in the lowest skill score quartile (Q1) would be as high as 32%, 39%, 45% and 55% for Tasks A–D, respectively, whereas the likelihood of selecting one in the highest skill score category would, be only 32%, 17%, 21% and 18%, for the same tasks. That is, a focus on low effort estimates in the selection process would be more likely to result in lower-skill (Q1) than higher-skill (Q4) software developers. Furthermore, skill indicators typically used in industry, e.g. selection of senior developers or those with long experienced as programmers using the relevant programming language, will not significantly improve the likelihood of selecting the most skilled, either. This suggests that typical provider and developer selection strategies in software development, e.g. emphasising low estimates/price and using background information typically provided in CVs, involve a high risk of selecting low-skill developers. Accordingly, other methods should be used in the selection process. More reliable means of selecting skilled developers or providers may include the use of trialsourcing, i.e. larger-scale skill evaluation based on relevant project tasks, as part of the selection process [1] or to use skill tests similar to those proposed in [16].

The observation that the effort estimates of some of the smaller tasks were better correlated with programming skill (up to 0.5) and had higher hit rates (up to 70%) implies that it may be possible to design estimation tasks suitable for identifying the best programmers. For instance, tasks requiring programming skill to identify that they are in fact simple are well suited for this purpose. More research should be conducted in this direction.

4.4 Threats to validity

Construct validity: True vs measured programming skill

The aim has been to measure the programming skill of the participants in the study and we have used a validated measurement tool for that purpose. All relevant aspects of programming skill of a developer, however, can clearly not be measured by the performance on five smaller programming tasks. What was achieved may therefore be described as a comparison of self-assessed programming skill, in the form of estimated effort required to complete a task, with more objective programming skill measures. Even though not all types of tasks and contexts are represented, the prior validation of the skill measurement tool suggests that the skill tests represent typical software development well. The consistency of the results, in particular the similarity relations between the skill scores and the estimates across all four larger tasks and across all five smaller tasks, also support the validity of the skill measures.

Internal validity: Causality vs correlation

The explanation of the findings is motivated by a belief in a causal, perhaps cyclic, relation between low skill in completing and a low skill in estimating the required effort for a software development task. Strictly speaking, analyses based on data from observational studies (as in the present case) will, however, only enable causal claims if all relevant variables are included, the type of model is appropriate and all model assumptions are met. This is not likely to be the case in the present study. An important reasons for believing that what we have reported is a causal, and not only correlational, relationship between lower programming and lower estimation skill is that the main results correspond to results in previous studies, or are natural consequences of that low competence in understanding a task can lead to failure to understand its simplicity (or complexity). This, of course, does not imply that there are no other variables (not included in the present analysis) that may explain parts of the results and moderate the proposed causal relations.

External validity: Generalising results to other software development contexts

The four larger software development tasks were selected to represent typical smaller software projects, and the five smaller tasks to represent core programming activities. Estimating and completing both types of software development tasks is likely to be common for the developers participating in this study. Even though the underlying mechanisms may be

present in other software development contexts, the effect size, e.g. the degree to which skill level will affect estimation over-optimism, is expected to be strongly context-dependent. For example, contexts with tasks whose complexity is even more hidden for those with lower, but not for those with higher skill, may increase the stronger over-optimism of the less skilled even more compared to what we observed.

This study did not request company-level, or group-based, effort estimates, but only estimates from individual developers. One may argue that companies hardly ever let those with low programming skill estimate software projects, and therefore less company-level estimation problems occurs. An examination of the data demonstrated that this was not the case. Developers responsible for the entirety of project estimates (n = 69) were as likely to belong to the lowest skill score category (23% with skill scores in Q1) as to the highest skill score category (23% with skill scores in Q4). It is nevertheless possible that the effect size will be lower in various settings, e.g. when several software professionals contribute to the estimation using appropriate group-based practices. Examining the same effects as those in this study in the context of group-based effort estimation is an interesting topic for further research.

5. Conclusion

The relations between effort estimates and programming skill (RQ1) and between skill indicators and measured programming skill (RQ2) were examined, with the following contributions:

- Documenting the relevance of the well-established Dunning–Kruger effect on the larger software development tasks, i.e. lower programming skill was connected with more *over-optimistic* estimates of one's own performance. Not only were the effort estimates more over-optimistic, but they were on average lower than those of the developers with the highest programming skill.
- Extending the Dunning–Kruger effect by observing that there are contexts in which lower skill is connected with more *over-pessimistic* effort estimates. We conjecture that the general effect of low skill is higher estimation uncertainty, and occasionally, this is connected with lack of skill in identifying the simplicity of a task, leading to over-pessimistic effort estimates.
- Replicating and extending previously not examined skill indicators results documenting that the connections between measured programming skill and programming skill indicators typically used in industry when selecting among developers or providers are weak.
- Documenting a Dunning-Kruger effect related to estimation skills. Those with the lowest skill in effort estimation over-estimated their estimation skill the most.

Practical consequences of our results may include:

- Software clients and managers who emphasise low effort estimates, or the derived lower price, when selecting software developers to complete larger tasks do not only run a risk of selecting over-optimistic developers (the winner's curse). They also run higher risk of selecting the least skilled developers. Assuming that the results generalise to larger-scale software development project contexts, this cautions against selecting the lowest bidder, particularly when objective skill tests of the provider and/or its software developers are not conducted.
- Skill indicators typically used in industry, such as those included in the developers' CV, the company assessed skill categories, the developers' confidence in the accuracy of their estimates and assessed problem-solving ability, are not reliable. To identify programming skill, other means are required, such as work sample testing and trialsourcing.

We observed that the best indicator of programming skill was the developers' effort estimates on relatively simple programming tasks, in which those with higher skill seemed to have been more able to identify the tasks' simplicity compared with those with lower skill. The

use of estimates on carefully designed programming tasks as indicators of skill, we argue, is an interesting topic for further research.

Note: The first and second author have financial interests in the company (Technebies) that owns the skill testing tool used in this study.

References

[1] Jørgensen, M., *Better selection of software providers through trialsourcing.* Ieee Software, 2016. **33**(5): p. 48-53.

[2] Anda, B.C.D., D.I.K. Sjøberg, and A. Mockus, *Variability and reproducibility in software engineering: A study of four companies that developed the same system.* IEEE Transactions of Software Engineering, 2009. **35**(3): p. 409-429.

[3] Kruger, J. and D. Dunning, *Unskilled and unaware of it: How difficulties in recognizing one's own incompetence lead to inflated self-assessments.* Journal of Personality and Social Psychology, 1999. **77**(6): p. 1121-1134.

[4] Simons, D.J., *Unskilled and optimistic: Overconfident predictions despite calibrated knowledge of relative skill.* Psychonomic Bulletin & Review, 2013. **20**(3): p. 601-607.

[5] Meeran, S., P. Goodwin, and B. Yalabik, *A parsimonious explanation of observed biases when forecasting one's own performance.* International Journal of Forecasting, 2016. **32**(1): p. 112-120.

[6] Krajc, M. and A. Ortmann, *Are the unskilled really that unaware? An alternative explanation.* Journal of Economic Psychology, 2008. **29**(5): p. 724-738.

[7] Pavel, S.R., M.F. Robertson, and B.T. Harrison, *The Dunning-Kruger effect and SIUC University's aviation students.* Journal of Aviation Technology and Engineering, 2012. **2**(1): p. 125-129.

[8] Fogarty, G.J. and D. Else, *Performance calibration in sport: Implications for self‐confidence and metacognitive biases.* International Journal of Sport and Exercise Psychology, 2005. **3**(1): p. 41-57.

[9] Dunning, D., C. Heath, and J.M. Suls, *Flawed self-assessment: Implications for health, education, and the workplace.* Psychological science in the public interest, 2004. **5**(3): p. 69-106.

[10] Jørgensen, M., *The influence of selection bias on effort overruns in software development projects.* Information and Software Technology, 2013. **55**(9): p. 1640-1650.

[11] Jørgensen, M. *Software development contracts: the impact of the provider's risk of financial loss on project success*. In *10th Int. Conference on Cooperative and Human Aspects of Software Engineering*. Buenos Aires. IEEE Press. p. 30-35.

[12] Løhre, E. and M. Jørgensen, *Numerical anchors and their strong effects on software development effort estimates.* Journal of Systems and Software, 2016. **116**: p. 49-56.

[13] Vierordt, K., *Der zeitsinn nach versuchen*. 1868: H. Laupp.

[14] Halkjelsvik, T. and M. Jørgensen, *From Origami to Software Development: A Review of Studies on Judgment-Based Predictions of Performance Time.* Psychological Bulletin, 2012. **138**(2): p. 238-271.

[15] Damisch, L., T. Mussweiler, and H. Plessner, *Olympic medals as fruits of comparison? Assimilation and contrast in sequential performance judgments.* Journal of Experimental Psychology: Applied, 2006. **12**(3): p. 166.

[16] Bergersen, G.R., D.I.K. Sjøberg, and T. Dybå, *Construction and validation of an instrument for measuring programming skill.* IEEE Transactions on Software Engineering, 2014. **40**(12): p. 1163-1184.

[17] Bergersen, G.R. and J.-E. Gustafsson, *Programming skill, knowledge, and working memory among professional software developers from an investment theory perspective.* Journal of Individual Differences, 2011. **32**(4): p. 201-209.

[18] Andrich, D., *A rating formulation for ordered response categories.* Psychometrika, 1978. **43**(4): p. 561-573.

[19] Mair, P. and R. Hatzinger, *CML based estimation of extended Rasch models with the eRm package in R.* Psychology Science, 49(1), 26-43., 2007. **49**(1): p. 26-43.

[20] Galwey, N.W., *Introduction to mixed modelling: beyond regression and analysis of variance*. 2014: John Wiley & Sons.

[21] Tu, Y.-K., I.H. Maddick, G.S. Griffiths, and M.S. Gilthorpe, *Mathematical coupling can undermine the statistical assessment of clinical research: illustrations from the treatment of guided tissue regeneration.* Journal of Dentistry, 2004. **32**(2): p. 133-142.

[22] Somers, R.H., *A new asymmetric measure of association for ordinal variables.* American sociological review, 1962: p. 799-811.

[23] Goodman, L.A. and W.H. Kruskal, *Measures of association for cross classifications.* Journal of the American statistical association, 1954. **49**(268): p. 732-764.

[24] Siegmund, J., C. Kästner, J. Liebig, S. Apel, and S. Hanenberg, *Measuring and modeling programming experience.* Empirical Software Engineering, 2014. **19**(5): p. 1299-1334.

[25] Bryan, G.E. *Not all programmers are created equal*. In *Aerospace Applications Conference*. 1994. IEEE. p. 55-62.
[26] Alicke, M.D. and C. Sedikides, *Handbook of self-enhancement and self-protection*. 2011: Guilford Press.
[27] Moløkken, K. and M. Jørgensen. *A review of surveys on software effort estimation*. In *International Symposium on Empirical Software Engineering*. 2003. Rome, Italy. IEEE. p. 223-230.
[28] Budzier, A. and B. Flyvbjerg, *Overspend? Late? Failure? What the data say about IT project risk in the public sector.* arXiv preprint arXiv:1304.4525, 2013.
[29] Zell, E. and Z. Krizan, *Do people have insight into their abilities? A metasynthesis.* Perspectives on Psychological Science, 2014. **9**(2): p. 111-125.
[30] Jørgensen, M., *Selection of strategies in judgment-based effort estimation.* Journal of Systems and Software, 2010. **83**(6): p. 1039-1050.
[31] Rush, C. and R. Roy, *Expert judgement in cost estimating: modelling the reasoning process.* Concurrent Engineering: Research and Applications, 2001. **9**(4): p. 271 - 284.
[32] Jazayeri, M. and M.N. Shadlen, *Temporal context calibrates interval timing.* Nature neuroscience, 2010. **13**(8): p. 1020.
[33] Hollingworth, H.L., *The central tendency of judgment.* The Journal of Philosophy, Psychology and Scientific Methods, 1910. **7**(17): p. 461-469.
[34] Tamrakar, R. and M. Jørgensen. *Does the use of Fibonacci numbers in Planning Poker affect effort estimates*. In *16th International Conference on Evaluation and Assessment in Software Engineering*. 2012. IET. p. 228-232.
[35] Kern, T., L.P. Willcocks, and E. Van Heck, *The winner's curse in IT outsourcing: Strategies for avoiding relational trauma.* California Management Review, 2002. **44**(2): p. 47-69.
[36] Jørgensen, M., *A survey on the characteristics of projects with success in delivering client benefits.* Information and Software Technology, 2016. **78**: p. 83-94.
[37] Jørgensen, M., *How to avoid selecting bids based on overoptimistic cost estimates.* IEEE Software, 2009. **26**(3).
[38] Jørgensen, M., P. Mohagheghi, and S. Grimstad, *Direct and indirect connections between type of contract and software project outcome.* International Journal of Project Management, 2017. **35**(8): p. 1573-1586.

Appendix

Table A1: Mixed model analysis results of the larger tasks

Fixed effects	**Variable**	**Categories**	**Coefficient**	**95% CI**
	Intercept		8.39	[8.15; 8.63]
	Task (D is the reference task)	A	-0.18	[-0.31; -0.04]
		B	0.59	[0.46; 0.73]
		C	-0.21	[-0.34; -0.07]
	Skill category (Q4 is the reference category)	Q1	-0.53	[-0.92; -0.14]
		Q2	0.26	[-0.12; 0.64]
		Q3	0.38	[-0.13; 0.90]
	Task x Skill category (D and Q4 are the references)	A x Q1	0.09	[-0.13; 0.31]
		A x Q2	-0.21	[-0.43; 0.00]
		A x Q3	0.08	[-0.20; 0.37]
		B x Q1	-0.25	[-0.47; -0.03]
		B x Q2	0.26	[0.05; 0.48]
		B x Q3	-0.09	[-0.38; 0.19]
		C x Q1	0.26	[0.04; 0.48]
		C x Q2	-0.07	[-0.29; 0.14]
		C x Q3	-0.09	[-0.33; 0.24]
Random effect	**Variable**	**Variance**	**Percent of total variance**	
	Developer	1.13	68%	
	Residual	0.53	32%	

Table A2: Mixed model analysis results of the smaller tasks

Fixed effects	**Variable**	**Values**	**Coefficient**	**95% CI**
	Intercept		5.03	[4.85; 5.22]
	Task (I is the reference task)	E	0.25	[0.09; 0.42]
		F	-0.05	[-0.19; 0.09]
		G	-0,80	[-0.94; -0.66]
		H	-0.43	[-0.57; -0.28]
	Skill category (Q4 is the reference categories)	Q1	0.59	[0.36; 0.83]
		Q2	-0.10	[-0.29; 0.09]
		Q3	-0.18	[-0.35; -0.02]
	Task x Skill category (I and Q4 are the references)	E x Q1	0.06	[-0.21; 0.33]
		E x Q2	0.17	[-0.22; 0.56]
		E x Q3	-0.02	[-0.26; 0.22]
		F x Q1	0.10	[-0.16; 0.37]
		F x Q2	-0.23	[-0.48; 0.02]
		F x Q3	0.22	[-0.03; 0.46]
		G x Q1	0.20	[-0.07; 0.46]
		G x Q2	-0.13	[-0.38; 0.01]
		G x Q3	-0.24	[-0.49; 0.01]
		H x Q1	0.05	[-0.20; 0.30]
		H x Q2	0.02	[-0.28; 0.32]
		H x Q3	0.03	[-0.19; 0.26]
Random effect	**Variable**	**Variance**	**Percent of total variance**	
	Developer	0.70	54%	
	Residual	0.60	46%	

Figure A1: Residuals for the mixed model of the larger tasks

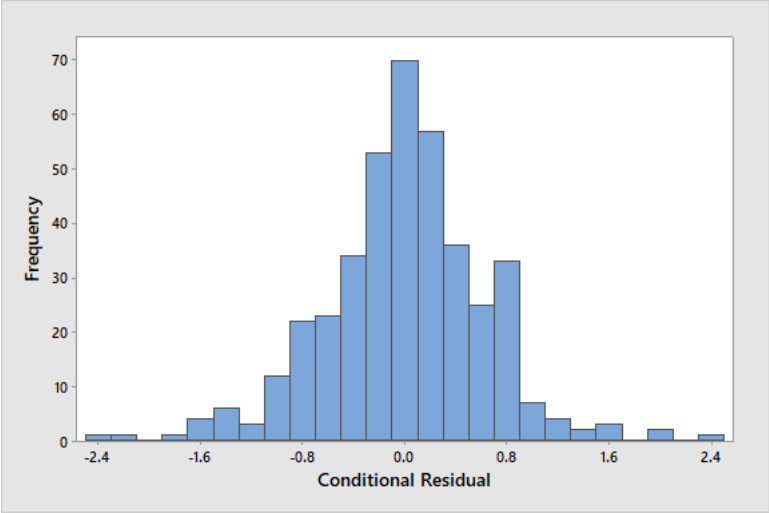


Figure A2: Residuals for the mixed model of the smaller tasks