Inferring Skill from Tests of Programming Performance: Combining Time and Quality

Gunnar R. Bergersen
Department of Informatics,
University of Oslo and Simula
Research Laboratory, Norway
gunnab@ifi.uio.no

Jo E. Hannay
Simula Research Laboratory,
P.O. Box 134, NO-1325 Lysaker,
Norway
johannay@simula.no

Dag I. K. Sjøberg
Department of Informatics,
University of Oslo, P.O. Box 1080,
NO-0316 Oslo, Norway
dagsj@ifi.uio.no

Tore Dybå
Department of Informatics,
University of Oslo and
SINTEF, Norway
tore.dyba@sintef.no

Amela Karahasanović
SINTEF and Department of Informatics,
University of Oslo
P.O. Box 124 Blindern, NO-0314 Oslo, Norway
amela@sintef.no

Abstract—**The skills of software developers is important to the success of software projects. Also, when studying the general effect of a tool or method, it is important to control for individual differences in skill. However, the way skill is assessed is often *ad hoc,* or based on unvalidated methods. According to established test theory, validated tests of skill should infer skill levels from well-defined performance measures on multiple, small, representative tasks. In this respect, we show how time and quality, which are often analyzed separately, can be combined as task performance and subsequently be aggregated as an approximation of skill. Our results show significant positive correlations between our proposed measures of skill and other variables, such as seniority, lines of code written, and self-evaluated expertise. The method for combining time and quality is a promising first step to measuring programming skill in both industry and research settings.**

Programming; skill; performance; time; quality; productivity

I. Introduction

The skills of individual software developers have a large impact on the success of software projects. Also, differences in programming performance reported in the late 1960s, indicate that the authors believed levels of performance varied dramatically. Although more recent research [13,32] is more conservative in their assertions, companies that succeed in hiring the best people will nevertheless achieve great economic and competitive benefits [17,34,37].

Individual differences in skill also affect the outcome of empirical studies. When evaluating alternative processes, methods, or tools, the effect of using a specific alternative may be mediated by skill levels. For example, in an experiment on the effect of a centralized versus delegated control style, the purportedly most-skilled developers performed better using a delegated control style than with a centralized one, while the less-skilled developers performed better with the centralized style [5]. In another experiment, skill had a moderating effect on the benefits of pair programming [4].

However, determining the skill level of software developers is far from a trivial task. In the work life, there are common-sense guidelines from experienced practitioners on how to distinguish the good from the bad [37]. But there seems to be consensus that this crucial human resource management task remains difficult. Often, job recruitment personnel use tests that purport to measure a variety of traits, such as general cognitive abilities (intelligence), values, interests, and measures of personality, to predict job performance [11]. Research has, however, established that work sample tests in combination with General Mental Ability (GMA) testing are among the best predictors of job performance [34]. GMA is a general aspect of intelligence and is best suited for predicting performance on entry-level jobs or job-training situations. By contrast, work sample tests are task-specific and are integrated closely with the concept of job skill [15]. Although the predictive validity of standardized work samples exceed that of GMA alone, these predictors seem to yield the best results when combined [34].

In the context of empirical studies in software engineering, the notion of programming skill is generally not well founded. This has led to studies that failed in adequately correcting for bias in quasi-experimental studies [23]. Often the more general concept of *programming expertise* is used, with little validation. For example, in a recent study [20], we conceptualized programming expertise as the level of seniority (junior, intermediate, senior) of the individual programmer as set by their manager. While bearing some relevance to the consultancy market, this conceptualization is not sufficient to capture the skill of individual programmers. The concepts of expertise and skill are also operationalized in questionable ways in other domains; see [22] for a survey of operationalizations in IT management.

The focus of this paper is as follows. Given a small set of programming tasks, how does one infer the candidates' programming skill from both the quality of the task solutions and the time spent performing the tasks? It is well recognized that the combination of task quality and time is essential to define skill [15,16], but how to combine them in practice is challenging. For example, how does one rank programmers who deliver high quality slowly, relative to those who deliver lesser quality more quickly? This paper addresses such challenges and proposes a method for combining quality and solution time into a single ordinal score of performance (i.e., *low*, *medium*, *high*). Multiple performance scores are then aggregated to form an ordinal approximation of programming skill. The method is demonstrated by using data from two existing experiments.

Section 2 gives the theoretical and analytical background for skill as a subdomain of expertise and discusses how quality and time are currently dealt with. Section 3 describes how time and quality were combined as programming performance on tasks. Section 4 reanalyzes two existing data sets according to the arguments given in the previous sections. Sections 5 and 6 discuss the results and conclude the paper.

II. Background

A. *Expertise*

Expertise is one of the classic concepts of social and behavioral science. Expertise is usually related to specific tasks within a given domain and does not in general transfer across domains or tasks [15,35]. Expertise has several aspects; we present five of these in Fig. 1(a). The aspects are all related. For example, in the usual descriptions of skill acquisition [1,14,16], which is a subdomain of expertise, an individual starts by acquiring declarative knowledge, which for experts is qualitatively different in representation and organization compared to novices [15,38]. Next, through practice, declarative knowledge is transformed into procedural skill, which at first is slow and error-prone [16]. However, though extended experience, performance improves and experts tend also to converge on their understanding of the domain in which they are an expert as well [35,36] (i.e., consensual agreement). Experts also regard themselves as being experts, for example, through the use of self-assessments. Ultimately, the desired effect of expertise is superior performance on the tasks in which one is an expert. In our context, this is performance on real-world programming tasks. However, predicting future job performance by observing actual job performance is unreliable and inefficient [11]. It is therefore desirable to design quick tests based on how well an individual reliably performs on representative tasks [15].

B. *Skill*

We generally understand skill as performance on small representative tasks. Note, though, that inferring skill from a reliable level of performance on representative tasks is not the same as defining it by performance on the job. Representative tasks in our context are those smaller tasks which merely represent real-world tasks, and for which there are well-defined measures of performance [15]. Additionally, such measures are typically regarded as situations of *maximum performance*, whereas behavior on the job would constitute *typical performance* [11]. Motivation plays a central role in predicting typical performance in a job situation (see [7] for an overview), whereas potential positive or negative consequences for a test-taker would affect a situation demanding maximum performance.

Generalizing from performance on small representative tasks to performance on the job requires an understanding of key mechanisms at play shared between tasks in the two settings. This is theory-driven generalization [33], based on the economy of artificiality [21]. In the absence of, or as a complement to, strong theory, it is useful to seek confirmation of how well skill measures coincide with other aspects of expertise. This is relevant for skill in programming.

Anderson et al. [1,2] investigated programming skill from a psychological perspective. They reported that both coding time and the number of programming errors decreased as skill improved. Further, programming in LISP required the learning of approximately 500 if-then rules. The acquisition of these rules followed a power-law learning curve: the improvement in performance was largest at first and then decelerated until an asymptote was reached. Thus, the relationship between amount of practice (extended experience) and performance was non-linear. However, if amount of practice and performance were logarithmically transformed, an approximately linear trend was observed. This phenomenon is widely observed and is often referred to as the *log-log law of practice* [31].

Fitts and Posner [16] have extensively studied skill acquisition. Within many different domains of expertise, they found that with increased skill, the number of errors in performance decreases and the speed with which a task is executed increases. Regarding measures of skill, they state:

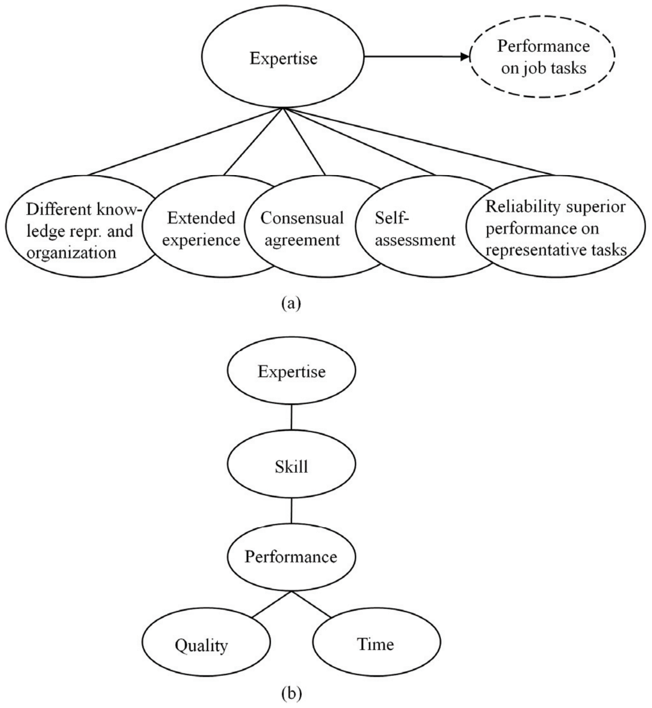

Figure 1 Expertise (a) and skill as one aspect of expertise (b) with relations to time and qualtiy as variables though the concept of performance. The desired effect of expertise is superior performance on job tasks.

"[t]he measure should take into account the length of time taken to perform a skill as well as the accuracy with which it is performed" [16, p. 85]. Therefore, time and quality (the latter being a generalization of accuracy) are intimately linked to skill, and the term performance is linked to all three concepts. Because skill affects performance [11], we can hierarchically structure the five concepts *expertise*, *skill*, *performance*, *time*, and *quality* as shown in Fig. 1(b). From the top, expertise, which should affect job performance, is a generalization of skill. Beneath, skill is inferred from performance on multiple tasks where reliably superior performance is a requirement. At the lowest level, time and quality, in combination, dictate whether programming performance overall is, say, low or high.

C. *Measures of programming performamce*

It is common in empirical software engineering to deal with quality and time separately when analyzing results; that is, one studies performance first in terms of quality and then in terms of time, often under the assumption that a solution meets some particular criterion for correctness (see, for example, [4,5]). We acknowledge that for many studies, this is acceptable. However, when the purpose is to characterize individual differences, problems may occur.

Time is a ratio variable with an inverse relation to performance (i.e., less time implies better performance). Quality, on the other hand, may consist of a plethora of variables where each one may have complex relations with each other and where all often cannot be optimized simultaneously [30]. Further, depending on how quality is operationalized, these variables may have different scale properties (i.e., nominal, ordinal, interval, or ratio). Therefore, when aiming to characterize individual differences, one may (a) disregard quality and report differences only in time spent or (b) only analyze time for observations surpassing some specific level of quality (often correctness), thereby adhering to the basic principle delineated by Thorndike and others in the 1920s: "the more quickly a person produces the *correct* response, the greater is his [ability]" [12, p. 440, emphasis added]. It is also possible to (c) devise acceptance tests that force everyone to work until an acceptable solution is achieved. Generally, we regard this as perhaps the most viable approach today, because variability in performance is expressed through time spent in total. However, by using (b) or (c), large portions of the dataset may be excluded from analysis, in particular when the proportion of correct solutions is low.

At the most fundamental level of the time/quality tradeoff problem, it is not clear how to place programmers who deliver high quality slowly relative to those who deliver lesser quality more quickly. In the datasets that are available to us, correctness and time are often negatively correlated. This indicates that the longer it takes to submit a solution, the lower is the likelihood of the solution being correct. Although this may seem contrary to what may be expected—that higher quality requires more time while lower quality requires less—there are two important distinctions to be made: first, there is a difference between quality in general and correctness specifically. Second, there is also a difference between *within-subject* and *between-subject* interpretations [10]; when a correct solution can be identified, a highly skilled individual can arrive at this solution in less time and with higher quality than a less capable individual (between-subject interpretation). But given more time, a single individual can generally improve an existing solution (within-subject interpretation).

Another challenge is identifying to what degree individual performance in a study is stable at a specific level, or high/low from one time to another. One way to address such concerns is to use multiple indicators of performance [6,18]. Based on the same principles for combining time and quality as performance delineated in this article, we have already advanced the measurement of skill using multiple indicators of performance [9]. However, a more detailed discussion of these principles involved is needed. It is to this discussion we will now turn.

D. *Using the Guttman structure for time and quality*

The two-by-two matrix in Fig. 2 has two possible values for quality (*low*, *high*) and two possible values for time (*slow*, *fast*). It is easy to agree that in this simplified example, "high performance" is represented by the upper right quadrant (fast and high quality) whereas "low performance" is represented by the lower left quadrant (slow and low quality). Further, it should also be straightforward to agree that the two remaining quadrants lie somewhere between these two extremes, say, "medium performance." However, which one of the two alternatives one would rate as superior, or whether they should be deemed equal, is a value judgment: in some instances, "fast and low quality" may be deemed superior to "slow and high quality." To address how performance should relate to different values for time and quality, we propose to use the principles delineated by Louis Guttman.

The Guttman scale was originally developed to determine whether a set of attitude statements is unidimensional [19]. In Guttman's sense, a perfect scale exists if a respondent who agrees with a certain statement also agrees with milder statements of the same attitude. The Guttman-structured scoring rules that we propose utilize the same underlying principle as the Guttman scale, although at a lower level of abstraction (a scale is a formal aggregation of indicators, whereas the structure we employ refers to the indicators themselves). The approach utilizes general principles as delineated by others [3], but which we have only addressed somewhat informally so far [8].

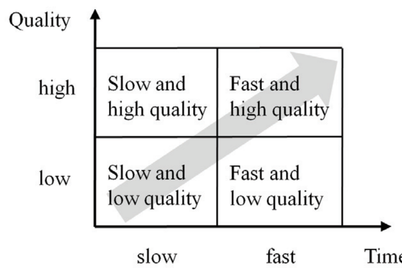

Figure 2. An example of scoring time and quality as performance based on value judgments.

With a Guttman structure it is possible to rank combinations of quality and time relative to each other as well as being explicit about how different tradeoffs in time and quality are scored. Performance on a programming task is thus determined by a series of well-ordered thresholds. Combined, these thresholds constitute a set of monotonically ordered response categories (i.e., an ordinal variable) in relation to performance. Surpassing a given threshold implies that all thresholds below it have been passed as well. For example, for a score of, say 3 (of 5 possible), the thresholds for obtaining scores of 0, 1, and 2 must have been passed, while the threshold for obtaining score 4 has failed. Quality can be deliberately emphasized over time (or vice versa) by adjusting score categories accordingly. A task that differentiates more on quality aspects may, further, be scored on multiple quality categories and a task that also differentiates more on time aspects may have more time categories.

III. Research Methods

This section describes how time and quality were combined as task performance using multiple indicators. Using two data sets, we show how scoring rules for tasks were operationalized and reanalyzed.

A. Data set 1

The first data set we reanalyzed is from a one-day study [5]. In the experiment, 99 consultants from eight software consultancy companies and 59 undergraduate and graduate students were paid to participate. The independent (treatment) variable in the experiment was the control style of the code (whether it is centralized or delegated). Five programming tasks were presented in succession to the subjects during the experiment. The first task i_1 (the pretest) was identical for both experimental groups, and the four next tasks, i_2–i_5 involved the independent variable. We analyze only the first four tasks here due to challenges in applying the last task to our purpose (see Section 4 C for why).

For a Guttman-structured scoring rule, we used the following approach for each of the tasks i_1–i_4: Let *Q1*, *T1*, *T2* and *T3* be dichotomous variables, scored as *requirement not met* = 0, *requirement met* = 1. Let *Q1* be functional correctness (as reported by the original authors), scored as *incorrect* = 0 or *correct* = 1. Let *T3* be time < 3rd quartile, *T2* be time < median, *T1* be time < 1st quartile. A Guttman structure for an ordinal performance score that applies to a single task combining quality and time is then defined by the Cartesian product *Q1×T3×T2×T1* as follows (*x* denotes either 0 or 1):

(0,*x*,*x*,*x*) = 0 (i.e., incorrect, time is irrelevant)
(1,0,*x*,*x*) = 1 (i.e., correct and very slow)
(1,1,0,*x*) = 2 (i.e., correct and slow)
(1,1,1,0) = 3 (i.e., correct and fast)
(1,1,1,1) = 4 (i.e., correct and very fast)

The matrix representation of this scoring rule is illustrated in Table 1(a). In using this structure, a solution must be correct before time is taken into consideration.

TABLE I. Score According to Time and Quality Thresholds

Score	***T3=0***	***T3=1***	***T2=1***	***T1=1***
Q1=1	1	2	3	4
Q1=0	0	0	0	0

(a) Dataset 1

Score	***T2=0***	***T2=1***	***T1=1***
Q2=1	2	3	4
Q1=1	1	1	1
Q1=0	0	0	0

(b) Dataset 2

Increasing scores for time are, further, only awarded in order (*T3* before *T2* and *T2* before *T1*). Additionally, the precedence of quality in this type of scoring rule reflects the view that, for this study, we do not consider a non-working solution to reflect high or medium performance, even when it is developed quickly.

We also constructed two alternative Guttman-based scoring rules to *Q1×T3×T2×T1* that differentiate less on time, but that are still based on the same *Q1* as above: *Q1×T2×T1* uses three categories for time based on the 33rd (*T2*) or 67th (*T1*) percentile; *Q1×T1* only uses two categories for time—above or below the median. The range of the overall performance score in all instances is equal to the number of dichotomous score variables plus one; for example, *Q1×T3×T2×T1* has one variable for quality and three for time, implying a total of five well ordered performance score categories with a range of 0–4.

The procedure described above was repeated for all four tasks. Because of different time distribution for each task, the quartiles and medians for time are calculated on a task-by-task basis. The resulting *score vector* consisted of four Guttman-structured score variables and the sum of these, the sum score, is the ordinal skill scale.

For comparison, we also devised two alternative scoring rules that combined quality and time for tasks by addition (*additive scoring rules*). On a task-by-task basis, we standardized quality and time (mean 0 and standard deviation 1) before adding the standardized variables as a composite score of performance. This implements treating “slow and high quality” as roughly equal to “fast and low quality” (as in Fig. 2), but where the continuous property of time is not forced into discrete categories. We name these scoring rules *Q+T* and *Q+lnT*. Here, time was negated in both instances, and for the latter variable, time was also logarithmically transformed before negation. Finally, we constructed scoring rules on the four quality variables alone (*Q*) and the four time variables alone (*T*).

It should be noted that the relationship between the score vector and the overall skill score is a many-to-one (surjective) function. For example, an individual with correct but very slow solutions for all four tasks when using *Q1×T3×T2×T1* receives the sum score of 4. An individual with a single correct solution with very fast time but the other three tasks incorrect would also receive the same sum score. Obviously, it is incorrect to characterize the latter instance as “reliably (superior) performance” because the individual exhibits superior performance on only a single task. We return to this issue in Section 3 C.

B. *Data set 2*

The second data set stems from three quasi-experiments which all used the same programming tasks. During a one-day study, the subjects were required to perform three different change tasks in a library application system of 3600 LOC, containing 26 Java classes. Two of the studies used students as subjects; one study used professionals. The study in [25] investigated the effects of different comprehension strategies using 38 subjects; the study in [24] compared feedback collection and think-aloud methods for 34 subjects; and the study in [27] studied the effects of expertise and strategies on program comprehension for 19 subjects. Additionally, the same pretest task as in Dataset 1 (i_1) was used. However, one of the studies had missing data for the last change task, thereby reducing the number of available tasks for analysis from four to three. Human graders scored the quality of each task on a five-point scale:

- 0: nothing done on the task (no code changes)
- 1: failure, does not compile or no discernible functional progress toward solution
- 2: functional anomalies, one or more subtasks are achieved
- 3: functionally correct, major visual anomalies
- 4: functionally correct, only minor cosmetic anomalies
- 5: functionally correct, visually correct, *i.e.* "perfect solution"

We defined a Guttman structure *Q1×Q2×Q3* for the dimension of quality as follows: The original scoring categories 0 and 1 should be collapsed into a single category, because neither might be preferred over the other. Thus, variable *Q1* was defined as "one or more subtasks achieved" (category 2 above). Next, the *Q2* variable was "functionally correct, but with major visual anomalies allowed" (category 3). Finally, we regarded the level of detail used for separating functionally correct with minor visual anomalies (category 4) and a "perfect solution" (category 5) as somewhat arbitrary; these two categories were therefore combined for *Q3*. For the time dimension, we used *T1×T2* to partition the time for those individuals who passed *Q3* into three groups. The matrix representation of this scoring rule, denoted *Q1×Q2×Q3×T1×T2*, is provided in Table 1(b).

We also devised alternative scoring rules using one and two dichotomous quality variables: *Q1×Q2×T2×T1* does not separate between major and minor visual anomalies when the solution is otherwise correct. And finally, *Q1×T2×T1* only separates between functionally correct solutions and those that are not functionally correct, with no attention given to visual anomalies. Finally, we devised scoring rules for *Q+T*, *Q+lnT*, *Q*, and *T* using the same procedure as in Dataset 1, but using three tasks instead of four.

C. *Analysis method and handling of missing data*

The analysis method for the two data sets, each using six different score operationalizations, included the same four basic steps. All time variables were negated (for *T*) or logarithmically transformed and then negated (for *Q+lnT*, *QlnT*) in order to increase interpretability so that high values indicate high performance.

1) *Using exploratory factor analysis.*

We extracted the main signal in the data for each scoring rule by Principal Component Analysis (PCA) using the analysis software PASW^{TM} 18.0. We used listwise deletion of missing variables, regression for calculating the factor score, and an unrotated (orthogonal) factor solution to maximize interpretability of each factor.

2) *Inspecting external and internal results*

Operationalizations of the scoring rules were compared with several experience variables. We report non-parametric correlations (Spearman's ρ, "rho") unless otherwise noted. We assumed that a valid scoring rule should correlate moderately and positively with relevant background variables such as developer category and length of experience. Because such variables are not influenced by our investigated score operationalizations, we refer to this analysis as *external results*.

Conversely, all the reported *internal results* are influenced by how each scoring rule was constructed. For internal results, we used the proportion of explained variance for the first Principal Component (PC), which is analogous to the sum score, as the signal-to-noise ratio for each scoring rule. Cronbach's α was used as an estimate of the internal consistency of the scores. To ascertain the applicability of each score operationalization, we used confirmatory factor analysis. We report the Root Mean Square Error of Approximation (RMSEA) using Amos^{TM} 18.0. RMSEA is a parsimony-adjusted index, as it favors models with fewer parameters. Further, RMSEA will be influenced negatively if level of performance is not consistent over multiple tasks (see Section 3 A). We used a tau-equivalent reflective measurement model with multiple indicators [29]. This implies that all tasks receive the same weight when calculating the sum score. All scoring rules are further regarded as ordinal scale approximations of skill.

3) *Handling of missing data*

Each dataset contain some missing data. For solutions that were not submitted, we applied the same basic principle as the authors of Dataset 2: "non-working solutions or no improvements in code" were equated with "nothing submitted at all" and scored as incorrect. Additionally Dataset 2 had some missing values for time. We did the same as the owners of this dataset and removed these observations altogether. Since missing data poses a threat to validity if data are not missing at random, we analyzed our results using data imputation as well. However, because we found that the same substantive results apply with or without data imputation, the results are reported without imputation.

IV. RESULTS

In this section, we first report the correlations between the investigated scoring rules and the subjects' background experience variables. Next, we report several indices that must be inspected together, such as explained variance, internal consistency and how well the scoring rules fit confirmatory factor analysis. Finally, we highlight some selected details about the scoring rules investigated.

A. External correlations

Table 2 shows correlations between experience variables and the proposed score operationalizations for both datasets. Developer category was only available for Dataset 1. In the initial classification scheme (i.e., *undergraduate* = 1, *intermediate* = 2, *junior* = 3, *intermediate* = 4, *senior* = 5) insignificant and low correlations were present between developer category and results (rho = 0.05–0.12). However, because many graduate students performed at levels comparable to seniors, it is questionable whether this operationalization of expertise is a monotonically increasing function of performance. When removing the two student categories (1 and 2) from the analysis, the company-assigned developer category complied with expectations to some extent: all correlations were significant and positive around 0.3.

The other experience variables were self-assessed. Years of programming experience (*lnProfExp*) is an aspect of extended experience. In general, the correlations for this variable were low and insignificant for all scoring alternatives, but were slightly improved having been logarithmically transformed (a justifiable transformation given the log-log law of practice). Java programming expertise (*SEJavaExp*) is a Likert scale variable ranging from *novice* = 1 to *expert* = 5. This variable was significantly and positively correlated around 0.3 with all of the scoring alternatives for Dataset 1. However, for Dataset 2, the correlations were lower and less systematic; caution should be shown when interpreting this result, however, due to the low number of observations (*n*). Overall, self-assessed Java programming expertise seems to have a non-linear but monotonically increasing relation to the proposed score operationalizations. Lines Of Code (*lnLOCJava*) written in Java is a self-estimated variable with positive skew and kurtosis, but approximates a normal distribution after logarithmic transformation. All scoring operationalizations were significantly and positively correlated with LOC (around rho = 0.3) with two exceptions: *Q* in Dataset 1 and *T* in Dataset 2.

B. Internal fit indices

Table 2 shows the fit indices of the investigated scoring rules. We used PCA to extract the main signal in the data, as represented by the first PC. The number of factors (*#f*) suggested by PCA indicates to what degree our expectations are present empirically. *Q* in Dataset 1 indicates a problem, because two factors are indicated by PCA.

The proportion of explained variance by the first PC (*%E*) indicates the signal-to-noise ratio for each score operationalization. The additive scoring rules (*Q*+*T*, *Q*+*lnT*) have the highest proportion of explained variance: logarithmic transformation before standardization of time produces additional explained variance.

Internal consistency is one way to investigate whether an individual's performance is stable over multiple tasks. High values for Cronbach's alpha (*α*) are better than low values (0.60 for group differences and 0.85 for individual differences are sometimes used). The two additive scoring rules do well in this respect, followed by Guttman-structured scoring rules. Further, *Q* in Dataset 1 and *T* in Dataset 2 have lower α than other alternatives.

Finally, we report how the different scoring alternatives fit according to confirmatory factor analysis. Lower RMSEA values signify better fit: values less than or equal to 0.05 indicate close approximate fit, values between 0.05 and 0.08 indicates reasonable error of approximation, and values above 0.10 suggest poor fit [26]. For Dataset 1, *Q1×T2×T1* and *Q1×T3×T2×T1* display reasonable model fit (< 0.08) whereas *T* and *Q* have poor fit. The additive scoring rules alternatives lie somewhere in between, and the logarithmically-transformed version (*Q*+*lnT)* has better overall fit than the untransformed version. *Q* for Dataset 2 has poor fit as well, even though this dataset has better overall confirmatory fit, despite the lower statistical power as can be seen by the wider 90% Confidence Intervals (CI).

TABLE II. CORRELATIONS AND CONFIRMATORY MODEL FIT OF SCORING ALTERNATIVES

	Non-parametric Correlations rho (n)				***Fit indices***			
Dataset 1	***Developer Category***	***lnProfExp***	***SEJavaExp***	***lnLOCJava***	***#f***	***%E***	***α***	***RMSEA [lo90, hi90]***
Q	(99) 0.26**	(157) 0.08	(158) 0.25**	(158) 0.12	2	33.3	0.45	0.145 [0.086, 0.211]
T	(93) 0.33**	(152) 0.16*	(152) 0.31**	(152) 0.38**	1	47.9	0.54	0.189 [0.131, 0.253]
Q+*T*	(93) 0.34**	(152) 0.14	(152) 0.30**	(152) 0.29**	1	48.7	0.65	0.096 [0.027, 0.166]
Q+*lnT*	(93) 0.35**	(152) 0.15	(152) 0.30**	(152) 0.29**	1	52.6	0.70	0.093 [0.021, 0.163]
Q1×T1	(99) 0.31**	(157) 0.07	(158) 0.33**	(158) 0.29**	1	45.0	0.58	0.094 [0.023, 0.164]
Q1×T2×T1	(99) 0.33**	(157) 0.11	(158) 0.31**	(158) 0.29**	1	47.7	0.63	0.076 [0.000, 0.149]
Q1×T3×T2×T1	(99) 0.35**	(157) 0.11	(158) 0.31**	(158) 0.30**	1	49.4	0.65	0.074 [0.000, 0.147]
Dataset 2								
Q	NA	(89) 0.12	(19) 0.14	(89) 0.36**	1	52.5	0.54	0.109 [0.000, 0.261]
T	NA	(89) –0.15	(19) –0.02	(89) 0.19	1	47.6	0.41	0.019 [0.000, 0.212]
Q+*T*	NA	(89) –0.01	(19) 0.10	(89) 0.35**	1	59.9	0.66	0.137 [0.000, 0.284]
Q+*lnT*	NA	(89) –0.02	(19) 0.10	(89) 0.34**	1	62.9	0.70	0.095 [0.000, 0.250]
Q1×T2×T1	NA	(89) 0.01	(19) 0.03	(89) 0.30**	1	52.6	0.55	0.000 [0.000, 0.179]
Q1×Q2×T2×T1	NA	(89) 0.05	(19) 0.23	(89) 0.34**	1	54.9	0.59	0.000 [0.000, 0.103]
Q1×Q2×Q3×T2×T1	NA	(89) 0.09	(19) 0.22	(89) 0.33**	1	55.7	0.60	0.000 [0.000, 0.153]

N is the number of observations, developer category is junior (3), intermediate (4) or senior (5), lnProfExp is the log-transformed number of years of professional programming experience where part time experience is counted as 25% of full time experience, SEJavaExp is self-evaluated Java programming expertise on a scale from novice (1) to expert (5), #f is the number of suggested factors by PCA, %E is percent total variance explained by the first PC, α is Cronbach's alpha, RMSEA is the Root Mean Square Error of Approximation with 90% low (lo90) and hi (hi90) confidence intervals. Data not available for analysis are marked NA. Correlations significant at the 0.05 level (two-tailed) are marked * and correlations significant at the 0.01 level are marked **.

Nevertheless, upon inspecting the lower CI of Dataset 1, there is sufficient statistical power to state that *T* fits poorly. However, as is evident by the upper CI of all alternatives, there is not sufficient statistical power to claim support for a close model fit for any of these alternatives either; *Q1×Q2×T2×T1* is overall the best fitting alternative with upper CI slightly above 0.10.

In summary, an analysis of *Q* and *T* separately seems problematic in terms of some correlations, relatively low explained variance, and problematic confirmatory fit in three out of four instances. The additive scoring rules show the highest levels of explained variance and internal consistency, but they display some problems with confirmatory model fit. Overall we found the best-fitting score operationalizations to be *Q1×T3×T2×T1* for Dataset1 and *Q1×Q2×T2×T1* for Dataset 2.

C. *Details for factors in Datasets 1 and 2*

Table 3 shows the correlations between all but one of the investigated scoring rules (*Q+lnT* was found to be a better alternative than *Q+T* and the latter is therefore not reported). Correlations below the diagonal are in terms of (non-parametric) Spearman's rho, which does not assume linearity between factors and may therefore be used. Parametric correlations (Pearson's *r*) are given above the diagonal for comparison. All correlations are significant at the 0.01 level (two-tailed).

In both datasets, *T* and *Q* have the lowest correlation with each other. For all Guttman-structured alternatives in Dataset 1, we may further observe how these scoring rules migrate from closeness with *Q* to closeness with *T* when additional time categories are added. Similarly, the Guttman-structured alternatives in Dataset 2 migrate from closeness with *T* to closeness with *Q* when additional quality categories are added. All proposed scoring alternatives also have more shared variance with *T* and *Q* separately, than *Q* and *T* have with each other. Further, the additive and Guttman-structured scoring alternatives are also somewhat similar in their rank ordering of individuals for Dataset 2 (rho > 0.6). For Dataset 1, in fact, they are highly similar in their rank ordering of individuals (rho > 0.9).

To verify that the scoring rules predict performance on other programming tasks, we used the *Q1×T2×T1* scoring rule of Dataset 1 to separate individuals into *low* and *high* skill groups. Using the results for task i_5, which is not a part of the investigated scoring rules, as a dependent variable and above/below mean sum score of tasks i_1–i_4 as the independent variable, we found that the high group performed much better than the low group: 67.1% had correct solutions for i_5 while the corresponding results for the low group was 27.8% correct (time could not be analyzed this way for i_5; see [5] for an explanation). We could further confirm that the high group had written significantly more LOC in Java and had more programming and Java experience as well. Moreover, by using three groups instead of two (i.e., *low*, *medium* and *high*) in terms of overall skill, similar results were obtained: the groups are well ordered according to external background variables, as well as on performance for i_5. For Dataset 2, we performed the same analysis, using the quality of task i_4 as the dependent variable and the sum score of i_1–i_3 as the independent variable. All correlations between the Guttman-structured scoring rules and the quality of i_4 were large (n = 52, rho = 0.51–0.53) and significant (p < 0.001).

We were also able to identify the treatment effect of Dataset 1. The dichotomous treatment variable was significantly and moderately correlated with the second PC of *Q1×T3×T2×T1* (rho = 0.42) and for the two additive scoring rules as well (rho = 0.36). Further, the treatment effect was not correlated with any of the first PC of the proposed scoring rules, suggesting that the effect of the treatment in this study is less than the individual variability. This implies that unless some degree of experimental control is available for individual variability—for example, through the use of pre-tests [23,33]—an experimenter would require many more subjects in a study to achieve the same statistical power. Perhaps worse, effects of practical importance might go undetected in the early phases of a research project.

TABLE III. CORRELATIONS FOR DATASET 1 AND 2

Scoring rule	***(1)***	***(2)***	***(3)***	***(4)***	***(5)***	***(6)***
Q (1)		0.36	0.76	0.82	0.75	0.71
T (2)	0.33		0.82	0.64	0.73	0.74
Q+lnT (3)	0.72	0.85		0.92	0.95	0.94
Q1×T1 (4)	0.80	0.70	0.92		0.96	0.96
Q1×T2×T1 (5)	0.75	0.78	0.96	0.95		0.98
Q1×T3×T2×T1 (6)	0.72	0.81	0.97	0.96	0.98	

(a) Dataset 1

Scoring rule	***(1)***	***(2)***	***(3)***	***(4)***	***(5)***	***(6)***
Q (1)		0.42	0.83	0.68	0.85	0.95
T (2)	0.41		0.83	0.53	0.56	0.50
Q+lnT (3)	0.84	0.80		0.70	0.83	0.85
Q1× T2×T1 (4)	0.64	0.49	0.62		0.86	0.75
Q1×Q2×T2×T1 (5)	0.81	0.52	0.79	0.83		0.89
Q1×Q2×Q3×T2×T1 (6)	0.96	0.45	0.84	0.70	0.87	

(b) Dataset 2

V. DISCUSSION

We begin by discussing the implications for research and practice when combining time and quality in empirical studies on programmers. Next, we discuss limitations and address how this work can be expanded in the future.

A. *Implications for research and practice*

In this reanalysis, we have presented a method for combining time and quality as an ordinal variable of performance. Results show that when programming performance on multiple tasks were aggregated, significant and positive correlations with skill could be obtained with several relevant expertise-related background variables. The strongest and most consistent correlations were obtained for LOC (around 0.3), which is highly similar to the value (of 0.29) reported in [9]. Seniority and self-evaluated expertise indicated more mixed results. For seniority in Dataset 1, statistical significant positive correlations could only be obtained when students were removed from analysis. For general programming experience, low and insignificant correlations were present. However, [9] reports a correlation of 0.29 between skill and months of Java programming experience; this may indicate that more precision (months instead of years) as well as specificity (*Java* experience as opposed to *general* programming experience) is required to yield higher correlations for this variable.

Nevertheless, using correlations as the only criteria for evaluating tests poses a problem; what the actual correlation is between a test score and a background variable will never be known for certain [10]. We therefore used confirmatory factor analysis to investigate whether performance on multiple tasks could be considered as "reliable (superior) performance" according to established literature on expertise and skill. We found unacceptable model fit in three of four instances when analyzing quality or time as separate variables. Furthermore, the only well-fitting variable (*T* in Dataset 2) did not accord with expectations of correlations with expertise background variables. Therefore, some concerns seem to exist when performance as a dependent variable is operationalized as time alone, or quality alone, in programming experiments. Such a problem will, however, remain undetected unless multiple tasks are present to be compared.

This study also demonstrates the importance of considering experiment constraints when analyzing individual variability of performance. For example, when loose time limits, or no limits, are used in a study and most subjects solve a task correctly, it is entirely plausible that, when analyzing correct solutions, between-subject variability is mostly present in the time variable. Conversely, if strict time limits are used and few subjects are able to finish on time, variability will mainly reside in the quality variable (as they would in, for example, an examination). Hence, the scoring of time and quality as a combined variable is dependent on the instrumentation as well as upon empirical results. This also implies that no universal scoring rule exists that applies equally well to all tasks in all situations.

For time as a variable, we generally obtained stronger and more consistent correlations when using non-parametric correlations and untransformed experience variables or parametric correlations with logarithmically-transformed experience. We concede that other transformations may be applicable as well, such as 1/time. Nevertheless, variables analyzed in this manner should almost always be plotted first and verified against theoretical expectations; there is often little theoretical rationale for expecting *a priori* that variables have a linear relation, even if each variable displays an approximate normal distribution. A similar caution should be observed when analyzing variables of quality. For example, for Dataset 2, it is problematic to assume that an increase in score from 2 to 3 (a difference of 1) amounts to the same increase as from 3 to 4; improving a correct solution from "major" to "minor visual anomalies" may require negligible time, whereas the improvement from one functionally correct subtask to a fully functionally correct solution may require substantial effort. Hence, quality variables should be treated as ordinal when in doubt and they are, further, most likely to have non-linear relations to other variables. Therefore, non-parametric correlations should be the first, not last, resort.

For the scoring rules that combine time and quality, we see two main competing alternatives in the present analysis: the additive scoring rules treated "slow and high quality" and "fast and low quality" (Fig. 2) as "medium performance." The Guttman-structured alternatives treated "fast and low quality" differently; quality had to be at an (operationally defined) acceptable level, before additional score points were awarded for time. Both alternatives demonstrated highly similar results in terms or correlations as well as rank order of individuals in terms of skill. Specifically, the two alternatives were highly similar in terms of the rank ordering of individuals in Dataset 1. It is therefore interesting to note that confirmatory model fit was much stronger in favor of the Guttman-structured scoring rules in Dataset 2 where less agreement between the Guttman-structured and additive scoring rules exist.

Some challenges are also present with using the additive scoring rules; partial scores of performance are awarded for fast solutions even though no improvement may be present. This implies that an individual who chooses not to participate seriously in a study will receive a higher score than an individual who seriously attempts all tasks, but fails.

Finally, given that some merit for the use of Guttman structures is established at this point, a practical issue arises around the number of score categories for time and quality. Because neither quality alone, nor time alone, fits the data particularly well in most instances, a sensible choice is to start at the extremes and work towards a compromise using the indices and recommendations reported here. Based on our experience, each score operationalization will usually display a peak of overall fit at some point; for example, when adding *Q3* to *Q1*×*Q2*×*T2*×*T1* in Dataset 2, confirmatory model fit began to decrease while the proportion of explained variance only marginally improved.

B. *Limitations*

The weaknesses in the present analysis have to do with the two sources of data as well as the methods we have used to analyze time and quality as an aggregated variable.

The experimental treatments in both datasets add noise to the data, which probably entails that all investigated scoring rules show worse fit than they would in data sets without such treatment. However, for this issue, it is at least true that the conditions were equal throughout our comparisons. There are also statistical independency problems between tasks. When a new task expands upon the solution of the previous task (i.e., a testlet-structure), this is still held to constitute one observation. Furthermore, performance in both studies is affected by motivational components; some peer pressure was present for most subjects in Dataset 1, whereas we believe motivation was more variable in Dataset 2. Finally, we used factor analysis on discrete ordinal variables as well as on dichotomous variables for quality. Optimally, one should use Bayesian estimation [28] or polychoric correlation matrices if the aim is to conclude more strongly. It is nevertheless somewhat common to see, for example, Likert scale variables analyzed using factor analysis, with a claim that the resulting factor score has interval-scale properties.

A potential objection to our confirmatory analysis using tau-equivalence is that relative weights for each task may be more appropriate for calculating the sum score. We investigated RMSEA from such a perspective as well (i.e., a congeneric factor model). Although slightly better fit could be obtained for all scoring alternatives, the same substantive results nevertheless continue to apply when comparing alternative ways of combining time and quality.

Other, perhaps, justifiable objections to our approach may be that multiple-factor (e.g., two rotated factors) or simplex-structured models should have been investigated. For Dataset 1, we have already done these analyses, but with inconclusive overall results and poor confirmatory fit. Therefore, we have chosen not to report these here. We could also have used non-linear regression or neural networks to maximize explained variance. However, by doing so, it would have been more difficult to interpret the results and the implied models for measurement would lack parsimony. Better, and more recent, models for measurement exist than the one we have used here. We now turn to such improvements where patterns in the score vector impose additional constraints on what scoring rules can be considered well-fitting overall.

C. *Reccomendations for future research*

In software engineering it is somewhat common to read about differences in programming performance ranging in an order of magnitude or more. However, as illustrated in Fig. 1, these differences pertain to performance *per se*, and not necessarily to individuals. For ratio comparisons of individuals, a quantitative scale is required in which units are separated by equal distances and where zero is well defined. This may be attempted though axiomatic measurement, which is measurement in a stricter sense than the putative measures discussed here. But it is unlikely that advances in this direction will happen in the near future (see [10] for present challenges). Nevertheless, we consider Rasch analysis [3] as a useful intermediary step for obtaining interval-scale approximations of programming skill. In this model, task difficulty is also an integral part of the measurement process, something the current model for measurement we employ (i.e., classical test theory) is lacking.

We have identified some additional directions for future work. For example, two or more internally consistent scoring perspectives based on the same data could be devised in which more score points are awarded either to quality or to time. By estimating skill for an individual based on these two approaches separately, it may be possible to make inferences about an individual's preferred working style during a test. If an individual is ranked higher on one score operationalization than the other, the relative difference may provide information about the value system an individual has for "fast and low quality" versus "slow and high quality" solutions. Further, different scoring rules on a task-by-task basis should be investigated as well. Although we applied the same scoring principle to all tasks in this study, there should be no *a priori* reason for requiring that a scoring rule must fit all tasks equally well to be valid.

Future work using Guttman-structured scoring rules for time and quality may also be directed at how to avoid the degradation of time as a continuous variable into discrete categories. However, any solution to this problem must somehow account for the likelihood that time will have skewed distributions. Although the additive scoring rule partially solves this problem by logarithmically transforming time, we would like to see more alternatives in the future that differ in how quality and time is combined as well as how measurement is conceptualized.

VI. Conclusion

In a reanalysis of two existing studies, we have shown through multiple score operationalizations how time and quality for programming task solutions may be meaningfully combined as an ordinal variable of performance. By aggregation of multiple performance variables into an ordinal skill score approximation, we obtained significant and positive correlations with relevant experience variables for all score alternatives that combined time and quality. Some degree of internal consistency was present, but only at a level high enough to roughly characterize group differences. We also showed that from correlations alone, it was difficult to choose what the so-called "best" scoring alternative might be. However, by using confirmatory factor analysis, we found some support for the Guttman-structured scoring rules over other alternatives, such as analyzing time or quality independently. Statistical power in our reanalysis was, however, not sufficient to conclude strongly, even though upper confidence intervals for some scoring alternatives almost indicate reasonable model fit for some alternatives.

This study implies that if time or quality is analyzed separately as a dependent variable, and experiment results are contrary to theoretical expectations, a reanalysis using the principles delineated here may be warranted. Threats to validity may further be present if the dependent variable of a study is not a monotonically increasing function of expertise or if multiple operationalizations of the same treatment do not indicate similar interpretations of the results. How individual variability in a study affects time, quality, or the two variables in combination also warrants closer scrutiny; instrumentation issues, such as time limits or task difficulty match with subject population, may also influence results in unexpected directions. Finally, we recommend the use of scoring rules that converge in terms of measurement while being parsimonious and interpretable.

Alternative score operationalizations in which time and quality are combined often were more similar to each other than they were to time or quality analyzed separately. Therefore, more attention is required to analyze how "performance" is operationalized in programming studies. We call for others to reanalyze their existing datasets where time and quality variables have been collected, and to score time and quality as a combined variable of performance. Such new datasets might even contain relevant background variables that can be further used to inform the strength of relations with performance and background variables, thereby making meta-analytic studies feasible in the future.

Acknowledgment

The authors are grateful to Erik Arisholm, Kaja Kværn, Annette Levine, Richard Thomas, and Gøril Tømmerberg for use of their data, and to Jan-Eric Gustafsson for detailed feedback on an earlier analysis of Dataset 1.

References

[1] J. R. Anderson, Acquisition of cognitive skill. Psychol. Rev., vol. 89, 1982, pp. 369–406.

[2] J. R. Anderson, F. G. Conrad, and A. T. Corbett, Skill acquisition and the LISP tutor. Cognitive Sci., vol. 13, 1989, pp. 467–505.

[3] D. Andrich, Rasch models for measurement. Beverly Hills, CA: Sage Publications, 1988.

[4] E. Arisholm, H. Gallis, T. Dybå, and D. I. K. Sjøberg, "Evaluating pair programming with respect to system complexity and programmer expertise," IEEE T. Software Eng., vol. 33, 2007, pp. 65–86.

[5] E. Arisholm and D. I. K. Sjøberg, Evaluating the effect of a delegated versus centralized control style on the maintainability of object-oriented software. IEEE T. Software Eng., vol. 30, 2004, pp. 521–534.

[6] A. Basilevsky, Statistical factor analysis and related methods: Theory and applications. New York: John Wiley & Sons, 1994.

[7] S. Beecham, N. Baddoo, T. Hall, H. Robinson, and H. Sharp, "Motivation in software engineering: A systematic literature review", J. Inform. Software Tech., vol. 50, 2008, pp. 860–878.

[8] G. R. Bergersen, "Combining time and correctness in the scoring of performance on items," Proc. Probabilistic Models for Measurement in Education, Psychology, Social Science and Health, Copenhagen Business School and the University of Copenhagen, Jun. 2010. http://tiny.cc/pl1i1.

[9] G. R. Bergersen and J.-E. Gustafsson. "Programming skill, knowledge and working memory among professional software developers from an investment theory perspective," J. Indiv. Diff, in press.

[10] D. Borsboom, Measuring the mind: Conceptual issues in contemporary psychometrics. New York: Cambridge University Press, 2005.

[11] J. P. Campbell, "Modeling the performance prediction problem in industrial and organizational psychology," in Handbook of industrial and organizational psychology, vol. 1, 2nd ed., M. D. Dunnette and L. M. Hough, Eds. Palo Alto, CA: Consulting Psychologists Press, 1990, pp. 687–732.

[12] J. B. Carroll, Human cognitive abilities: A survey of factor-analytic studies. Cambridge: Cambridge University Press, 1993.

[13] T. DeMarco and T. Lister, Peopleware: Productive projects and teams, 2nd ed. New York: Dorset House Publishing Company, 1999.

[14] H. L. Dreyfus, S. E. Dreyfus, and T. Athanasiou, Mind over machine: The power of human intuition and expertise in the era of the computer. New York: The Free Press, 1988.

[15] K. A. Ericsson, N. Charness, P. J. Feltovich, and R. R. Hoffman, Eds., The Cambridge handbook of expertise and expert performance. Cambridge: Cambridge University Press, 2006.

[16] P. M. Fitts and M. I. Posner, Human performance. Belmont, CA: Brooks/Cole, 1967.

[17] R. L. Glass, Facts and fallacies of software engineering. Boston: Addison-Wesley Professional, 2003.

[18] R. L. Gorsuch, Factor analysis, 2nd ed. Hillsdale, NJ: Lawrence Erlbaum Associates, 1983.

[19] L. L. Guttman, "The basis for scalogram analysis," in Scaling: A sourcebook for behavioral scientists, G. M. Maranell, Ed. Chicago: Aldine Pub. Co., 2007, pp. 142–171.

[20] J. E. Hannay, E. Arisholm, H. Engvik, and D. I. K. Sjøberg, "Personality and pair programming," IEEE T. Software Eng., vol. 36, 2010, pp. 61–80.

[21] J. E. Hannay and M. Jørgensen, "The role of deliberate artificial design elements in software engineering experiments," IEEE T. Software Eng., vol. 34, 2008, pp. 242–259.

[22] T. Hærem, Task complexity and expertise as determinants of task perceptions and performance: Why technology structure research has been unreliable and inconclusive. PhD diss., Norwegian School of Management BI, 2002.

[23] V. B. Kampenes, T. Dybå, J. E. Hannay, and D. I. K. Sjøberg, "A systematic review of quasi-experiments in software engineering," Inform. Software Tech., vol. 51, 2009, pp. 71–82.

[24] A. Karahasanović, A. K. Levine, and R. C. Thomas, "Comprehension strategies and difficulties in maintaining object-oriented systems: An explorative study," J. Syst. Software, vol. 80, 2007, pp. 1541–1559.

[25] A. Karahasanović and R. C. Thomas, "Difficulties experienced by students in maintaining object-oriented systems: An empirical study," Proc. Ninth Australasian Computing Education Conference (ACE2007), Australian Computer Society, 2007, pp. 81–87.

[26] R. B. Kline, Principles and practice of structural equation modeling, 2nd ed. New York: Guilford Press, 2005.

[27] K. Kværn, Effects of expertise and strategies on program comprehension in maintenance of object-oriented systems: A controlled experiment with professional developers. Master thesis, Department of Informatics, University of Oslo, 2006.

[28] S.-Y. Lee, Structural Equation Modeling: A Bayesian approach. Chichester: John Wiley, 2007.

[29] J. C. Loehlin, Latent variable models: An introduction to factor, path, and structural equation analysis, 4th ed. Mahwah, NJ: Lawrence Erlbaum Associates, 2004.

[30] J. McCall, "Quality factors," in Encyclopedia of software engineering, vol. 2, J. J. Marciniak, Ed., 1994, pp. 958–969.

[31] A. Newell and P. Rosenbloom, "Mechanisms of skill acquisition and the law of practice," in Cognitive skills and their acquisition, J. R. Anderson, Ed. Hillsdale, NJ: Erlbaum, 1981, pp. 1–56.

[32] L. Prechelt, "The 28:1 Grant/Sackman legend is misleading, or: How large is interpersonal variation really?", University of Karlsruhe, Technical report, 18, 1999.

[33] W. R. Shadish, T. D. Cook, and D. T. Campbell, Experimental and quasi-experimental designs for generalized causal inference. Boston: Houghton Mifflin, 2002.

[34] F. L. Schmidt and J. E. Hunter, "The validity and utility of selection methods in personnel psychology: Practical and theoretical implications of 85 years of research findings," Psychol. Bull., vol. 124, 1998, pp. 262–274.

[35] J. Shanteau, "Competence in experts: The role of task characteristics," Organ. Behav. Hum. Dec., vol. 53, 1992, pp. 252–266.

[36] J. Shanteau, D. J. Weiss, R. P. Thomas, and J. C. Pounds, "Performance-based assessment of expertise: How to decide if someone is an expert or not," Eur. J. Oper. Res., vol. 136, 2002, pp. 253–263.

[37] J. Spolsky, Smart and gets things done: Joel Spolsky's concise guide to finding the best technical talent. Berkeley, CA: Apress, 2007.

[38] S. Wiedenbeck, V. Fix, and J. Scholtz, "Characteristics of the mental representations of novice and expert programmers: An empirical study," Int. J. Man-Machine Studies, vol. 39, 1993, pp. 793–812.