Evaluating Methods and Technologies in Software Engineering with Respect to Developers' Skill Level

Gunnar R. Bergersen and Dag I. K. Sjøberg
Department of Informatics, University of Oslo,
P.O. Box 1080, NO-0316 Oslo, Norway
{gunnab, dagsj}@ifi.uio.no

Abstract—**Background:** It is trivial that the usefulness of a technology depends on the skill of the user. Several studies have reported an interaction between skill levels and different technologies, but the effect of skill is, for the most part, ignored in empirical, human-centric studies in software engineering. **Aim:** This paper investigates the usefulness of a technology as a function of skill. **Method:** An experiment that used students as subjects found recursive implementations to be easier to debug correctly than iterative implementations. We replicated the experiment by hiring 65 professional developers from nine companies in eight countries. In addition to the debugging tasks, performance on 17 other programming tasks was collected and analyzed using a measurement model that expressed the effect of treatment as a function of skill. **Results:** The hypotheses of the original study were confirmed only for the low-skilled subjects in our replication. Conversely, the high-skilled subjects correctly debugged the iterative implementations faster than the recursive ones, while the difference between correct and incorrect solutions for both treatments was negligible. We also found that the effect of skill (odds ratio = 9.4) was much larger than the effect of the treatment (odds ratio = 1.5). **Conclusions:** Claiming that a technology is better than another is problematic without taking skill levels into account. Better ways to assess skills as an integral part of technology evaluation are required.

Keywords: programming skill, pretest, experimental control, debugging, performance, replication

I. INTRODUCTION

When studying the effects of software processes, products, or resources, a researcher is often forced to keep constant or control for factors that may influence the outcome of the experiment. Because previous studies have shown large variability in programming performance, it is important to control for this. However, it is not a simple task to control for programming skill [9, 18, 29, 42], which is one of several factors that affect programming performance [13, 15].

Individual differences may also mediate the claimed benefit of different technologies. In a one-day experiment on professionals maintaining two different implementations of the same system, seniority had an effect on which system was better: The system that used a "poor" object-oriented design was better for juniors, whereas the system that used a "good" design was better for seniors [7]. The effect of pair programming on the same system was also investigated in [6]; overall, the juniors benefitted from working in pairs whereas the seniors did not. Such results are clearly problematic if one aims to generalize from the study population to a target population specified only as "software developers."

When an independent variable, such as skill or seniority, is correlated with the dependent (outcome) variable of a study, it is relevant to address this variable in relation to the experimental results [40]. Improved control can be achieved during experiment design (e.g., through blocking or matching) or in analysis (e.g., as a covariate). In both instances, the statistical power increases [32].

Another way to increase statistical power in studies is to reduce subject variability [40]. However, the individual differences of developers are, perhaps, some of the largest factors that contribute to the success or failure of software development in general [19, 27]. Several studies report an "individual-differences" factor (e.g., due to differences in skill) that is highly variable across individuals [20], teams [35], companies [1], and universities [30], thereby complicating analysis and adding uncertainty to the results. Meta-analysis has also confirmed that individual variability in programming is large, even though it may appear less than the 1:28 differences reported in the early days of software engineering [36]. Nevertheless, large variability in skill levels implies that one should be meticulous when defining the sample population as well as the target population in empirical studies in software engineering.

An indicator of programming skill that is easy to collect is months of experience or lines of code written by the subjects. Large meta-analyses have indicated that biographical measures, such as experience, generally have low predictive validity in studies on job performance [39]. At the same time, work sample tests that involve actual (job) tasks have the highest degree of validity.

Using one set of tasks to predict performance on another set of tasks is not new: Anderson studied the acquisition of skills in LISP programming [2] and found that "the best predictor of individual differences in errors on problems that involved one LISP concept was number of errors on other problems that involved different concepts" (p. 203). Therefore, pretests appear to be better measures of skill than biographical variables, even though they require a lot more instrumentation.

Calls for better pretests for programmers can be traced back to at least 1980 [see 18]. Yet, in a 2009 literature review on quasi experiments in software engineering, only 42% of the reported 113 studies applied controls to account for potential selection bias [29]. Among the studies that applied controls, only three experiments involved actual pretest tasks. (The remaining studies used covariates such as lines of code, exam scores, or years of experience.) The authors therefore restated previous calls [see 9, 18] for initiatives where the interaction between different types of technologies and software developer capabilities could be investigated.

This article reports a replication of a debugging study where a measure of programming skill is available using a pretest. Unlike [6, 7], where a *small* pretest and a comprehensive experiment were used, we conducted a *comprehensive* pretest and a small replication. Our overall research question is: what are the moderating effects of skill levels on the purported benefit of different technologies or methods? In this study, we investigated whether the usefulness of recursive implementations in debugging is invariant of skill level. Further, we also aimed to assess the individual variability in skill, which is potentially a confounding factor, with respect to different implementations of two small debugging tasks.

To do so, we analyzed the effect of using different debugging implementations in a specific measurement model (the Rasch model) where the effect of treatment is expressed as a function of skill. Moreover, we use professional software developers, thereby addressing the common criticism that researchers habitually use students in experiments [see, e.g., 9, 18, 42]. Although debugging studies and replications are interesting in their own right, the focus here is on methodical issues: Specifically, we investigate the effect of skill levels on the generalizability of the main conclusions of two earlier studies.

Section 2 describes method and materials. Section 3 reports results and section 4 analyzes these results using programming skill as a covariate. Section 5 discusses implications, limitations, and suggestions for further work. Section 6 concludes the study.

II. Method and Materials

Section 2.1 describes the material, experimental procedure and results of the original study. Section 2.2 describes the material and experimental procedure of our replication. Section 2.3 introduces the Rasch measurement model, which is used in the analysis in Section 4.

A. The original study

The original study involved 266 students who took a course on data structures [11]. The students were presented with two C implementation tasks: (1) a small (< 25 lines of code) search program for a linked list ("Find" task) and (2) a linked list that was to be copied ("Copy" task). Each task had either a recursive or iterative implementation (the treatment of the study). Both tasks contained a bug that had to be correctly *identified* and *corrected*. The study found that significantly more subjects identified the bug for the recursive versions than they did for the iterative version. A similar result was also found for one of the tasks in an earlier comprehension study using PASCAL [10]. Regarding correcting the bug, recursion also gave significantly better results with respect to the proportion of correct solutions for the Copy task (p = 0.019). However, the results for the Find task were in weak (non-significant) favor of iteration. When the results of the two tasks were combined, the recursive versions had 4.1% more correct solutions overall, a result that was not significant (p = 0.311). For the time required to correctly debug the tasks, the original study was not significantly in favor of any of the treatments.

The original study used a randomized within-subject (repeated measures) crossover design. Both treatments were presented to all subjects. Either one of them used the iterative treatment first and the recursive treatment second or vice versa. However, the Find task was always presented before the Copy task. Therefore, it is unknown to what extent an *ordering effect* is present [see generally 40], for example, whether iterative Find and then recursive Copy is an easier order to solve the tasks than recursive Find and then iterative Copy. The tasks were debugged manually using "hand tracing".

B. This replication

Our replication is part of an ongoing work for constructing an instrument for assessing programming skill. We conducted a study with sixty-five professional software developers who were hired from nine companies for approximately €40 000. The companies were located in eight different Central or Eastern-European countries. All the subjects were required to have at least six months of recent programming experience in Java. The subjects used the same development tools that they normally used in their jobs. The programming tasks, which included code and descriptions, were downloaded from an experiment support environment [8] that was responsible for tracking the time spent on implementing each solution. Neither the developers nor their respective companies were given individual results.

The study lasted two days and consisted of 17 Java programming tasks in total. A subset of 12 of these tasks had previously been found to adequately represent a programming skill as a single measure that is normally distributed and sufficiently reliable to characterize individual differences [13]. Further, this measure was also significantly positively correlated with programming experience, a commercially available test of programming knowledge, and several tests of working memory, which is an important psychological variable in relation to skill [see 44, 47]. The overall results accorded with a previous meta-analysis on job performance [see 39] and Cattell's investment theory, which describes how the development of skills in general is mediated by the acquisition of knowledge [see 22].

In this replication, the subjects received the two debugging tasks described above in addition to the 17 Java programming tasks. Allocation to treatment version (recursive or iterative) for both tasks was random. One subject was removed because the subject was an extremely low outlier regarding skill.

This resulted in 64 pairs of Find and Copy tasks in a crossover design, as shown in Table I. To reduce the risk of an ordering effect [see, e.g., 40], we improved the design of the original study by randomizing on task order (Find versus Copy first) and including the recursive-recursive and iterative-iterative designs. Further, all the 19 tasks were

TABLE I. THE DESIGN OF THE REPLICATED STUDY

Find[a]		Copy[a]		*n*
Recursive	***Iterative***	***Recursive***	***Iterative***	
X			X	22
	X	X		21
X		X		9
	X		X	12

[a] which of the two tasks were presented first was randomized

allocated to the subjects in random order on a subject-by-subject basis.

The subjects were given 10 minutes to solve each debugging task. They were also allowed three additional minutes to upload the solution. Up to five minutes were allowed for reading the task description prior to downloading the code task. It was explicitly explained to the subjects that the time they spent reading task descriptions was not included in the time recorded for solving the tasks. Tasks that were submitted too late (i.e., more than 13 minutes in total) were scored as incorrect. This procedure was explained to the subjects prior to the start of the study. Time was only analyzed and reported for correct solutions.

In our replication, we focus only on differences for whether a bug was *corrected* or not. Our study design and available resources did not enable us to identify whether a bug was correctly identified (see Section 2.1) and then incorrectly corrected. The time to correctly debug a task in our study was not comparable to the original study because of differences in how the tasks were presented to the subjects.

We used R [37] for statistical analysis. Unless otherwise noted, Fisher's exact test was used to test differences in correctness, Welch's t-test for differences in time, and Spearman's rho to report (non-parametric) correlations. A common feature of all these statistics is that they do not make strong assumptions about the distribution of the data. We use Fisher's test, which can report exact probabilities, in the presence of few observations rather than the Chi-squared differences test that calculates approximate *p*-values. Welch's t-test is similar to the Student's t-test, but it does not assume that the compared variables have equal variance.

We use two-tailed tests for differences when reporting *p*-values. For standardized effect sizes, we use Cohen's *d* and follow the behavioral science conventions in [25] when reporting the magnitude of an effect (see [28] for software engineering conventions). We use [24] for effect size conversions from odds ratio (OR) to *d* and report arithmetic means.

C. *The Rasch measurement model*

A measurement model explicates how measurement is conceptualized. Within psychological testing, the choice of measurement model establishes how abilities, such as intelligence or skills, are related to a person's responses on items [33]. An *item* is a generic term for any question, essay, task, or other formulated problem presented to an individual to elicit a response. The choice of measurement model dictates how patterns in responses to items should and should not appear. Failure to detect expected patterns and the presence of unwanted patterns may invalidate a researcher's claims to what is measured by a psychological test [4].

The original Rasch model [38] was published in 1960 and conceptualizes measurement of abilities according to a probabilistic framework. The model has similarities to conditional logistic regression and is sometimes referred to as a one-parameter Item Response Theory (IRT) model (see e.g., [33, 34]). The use of IRT models has increased in last half century. Nowadays, IRT models are central to large, multi-national testing frameworks, such as the PISA test [16], which is used to measure educational achievement of students across approximately 40 OECD countries.

The Rasch model belongs to a class of models that assumes unidimensionality, that is, the investigated ability can be represented by a single numerical value [4, 33]. Central to the Rasch model is the invariant estimation of abilities and item difficulties [4]. This is consistent with the general test-theory requirements set forth by pioneers in psychology nearly a century ago [see 46].

The original Rasch model only permits two score categories when a person solves a task: *incorrect* = 0 or *correct* = 1. Therefore, it is called the *dichotomous* Rasch model. In this model, the probability of a person with skill β to correctly answer a task with difficulty δ can be expressed as

$$\Pr = \frac{e^{\beta-\delta}}{1+e^{\beta-\delta}}. \quad (1)$$

The parameters β and δ are represented in log odds (i.e., *logits*). When β equals δ, the probability for a correct response is 0.50. The relative distance between skill and task difficulty follows a logistic (sigmoid) function that is S-shaped.

A generalization of the dichotomous Rasch model, derived by Andrich [3], allows the use of *more than two* score categories. It is therefore called the *polytomous* Rasch model. Although this model is more complex to express mathematically than the dichotomous model shown above (1), the general principles are the same for both models.

Even though the Rasch model uses discrete score categories, continuous variables such as time have previously been adapted to the scoring structure of the polytomous Rasch model. In [12, 15], we explicated the requirements for including programming tasks that vary in both time and quality dimensions simultaneously in the polytomous Rasch model. We now use the same model to express the effect of recursive versus iterative treatments for the two debugging tasks conditional on skill.

The Rasch analysis was conducted using the Rumm2020 software package [5]. A difference of *x* logits has uniform implications over the whole scale and is equal to an OR of e^x.

III. Results

Table II shows the proportion of correct solutions in the original study and the replicated one. There are three clear differences. First, the professionals in the replicated study clearly have a larger proportion of correct solutions than the students of the original study when comparing the mean of both tasks combined for the two studies (OR = 7.6, $d = 1.12$, $p < 0.001$).

Second, the probability of a correct solution for the Find task was higher in both studies (i.e., it is an *easier* task). For our replication, the difference in mean correctness between Find and Copy is large as well (OR = 4.5, $d = 0.83$, $p = 0.001$).

Third, our prediction that recursion would have a larger proportion of correct responses was disproved on a task-by-task basis compared with the original study. Our study supports the conclusion of the original study only for the recursive Find task: we found a larger proportion of correct solutions (OR = 6.5, $d = 1.03$, $p = 0.106$) with a 95% confidence interval (95CI) for the OR that ranges from 0.72 to 316. However, for the Copy task, the result was in favor of iteration because the odds ratio is less than 1 (OR = 0.94, 95CI = [0.30, 3.0], $d = -0.03$, $p = 1.0$).

In the original study, the mean time required to fix the debugging problem correctly yielded mixed and not significant results that were slightly in favor of recursion. (Mean time was in favor of recursion for the Find task and in favor of iteration for the Copy task.) In this replication, the mean time, which were measured in minutes, were not in favor of any of the treatments (recursive Find = 5.48, SD = 2.48; iterative Find = 5.65, SD = 2.82; recursive Copy = 6.30, SD = 2.58; iterative Copy = 5.95, SD = 2.68). However, the mean does not adequately represent the central tendency of the data in the presence of outliers or when the distribution is skewed.

Figure 1 shows boxplots of the time for correct solutions. As indicated by the whiskers, the spread of the data is somewhat wider for both iterative treatments. Further, the median is much closer to the first quartile than the third quartile, which indicates a positively skewed distribution. The original study only reported mean differences for time without referring to the distribution, standard deviation or median values; it is therefore unclear whether the mean is a good representation of the central tendency of their data or not.

TABLE II. PROPORTION OF CORRECT SOLUTIONS IN BOTH STUDIES

Task	*Treatment*	*Original study (n)*	*Replicated study (n)*
Find	Recursive	34.1% (*132*)	96.8% (*31*)
Find	Iterative	38.1% (*134*)	81.8% (*33*)
Copy	Recursive	29.9% (*134*)	63.3% (*30*)
Copy	Iterative	17.6% (*131*)	64.7% (*34*)
Mean of Find	*Both*	*36.1% (266)*	*89.1% (64)*
Mean of Copy	*Both*	*23.8% (265)*	*64.0% (64)*
Mean of both tasks	*Both*	*30.0% (531)*	*76.6% (128)*

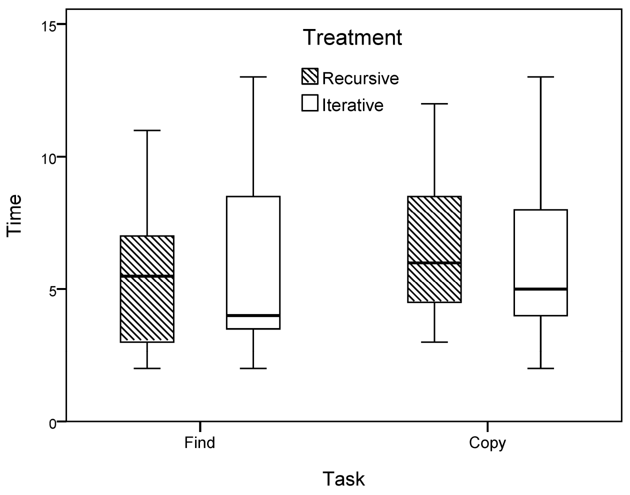

Figure 1. Distribution of time for correct solutions in the replication.

Within-subject designs permit pair-wise comparisons that may limit the confounding effect of individual variability [40]. In our replication, the recursive-recursive and iterative-iterative designs (Table I) cannot be used in such an analysis, because the same treatment is used for both tasks. Nevertheless, two-thirds of our subjects ($n = 43$) were given both treatments using a randomized crossover design. There are two relevant outcomes to analyze: those who had a correct recursive and incorrect iterative solution (i.e., in *favor of recursion*) and vice versa (i.e., in *favor of iteration*). Eleven subjects displayed results in favor of recursion and six subjects in favor of iteration. (A total of 26 subjects performed identically across treatments and are thus excluded from this analysis.) A null hypothesis of "equal probability in favor of either treatment" (i.e., in favor of recursion = in favor of iteration = 0.5) can then be tested against the alternative hypothesis of a higher probability in favor of recursion, using a binomial test. However, 11 successes in favor of recursion over 17 trials (11 + 6) give a p-value of only 0.167. Hence, the null hypothesis could not be falsified. Only weak support in favor of recursion was therefore present.

We have previously shown that the proportion of correct responses is higher for Find than for Copy (Table II). It is therefore interesting that 15 of the same 17 individuals who were in favor of one of the treatments were also in favor of the same treatment that they received for the (easiest) Find task. A null hypothesis of "no differences between tasks" using a binominal distribution could be falsified ($p = 0.001$). Hence, there is support for that the effect of differences in the difficulty between tasks appears larger than the effect of treatment across both tasks in an analysis using pair-wise comparisons.

In summary, only for the Find task, our replication supported the overall finding of the original study that recursion is associated with a larger proportion of correct answers. This result is contrary to the findings of the original authors who only found a significant difference in favor of recursion for the Copy task.

IV. RESULTS USING RASCH MODEL ANALYSIS

This section expands upon the previous section by including results for skill differences (Section 2.2). Section 4.1 gives justification as to why the measures of skill should be included in the analysis. Section 4.2 addresses to what extent random assignment to treatment was successful in our replication. Finally, while Sections 4.1 and 4.2 treat skill estimates as a "black box", Section 4.3 shows how the preference for iterative or recursive debugging tasks changes when the results (Section 3) are reanalyzed as a function of skill.

A. *Justification for using skill in the analysis*

In order to include a covariate such as skill in an analysis, the covariate must be correlated with the outcome of the experiment [40]. Table III shows the correlations between skill and the dual experiment outcomes of correctness and time to correctly debug the two tasks irrespective of treatment. All four correlations were large and significant with *p*-values below 0.003.

Correlations alone do not illustrate to what extent differences in skill accounts for practical differences in debugging performance on the two tasks. We therefore split the study population into two groups: The more skilled (variable *MoreSkill*) and the less skilled (variable *LessSkill*) groups consist of individuals with skill above and below the mean respectively. For the proportion of correct solutions for the Find task irrespective of treatment, *MoreSkill* had all tasks correct (100%), whereas *LessSkill* had 75.8% correct (OR cannot be computed, 95CI = [2.0, ∞], $p = 0.003$). For the Copy task, *MoreSkill* had a higher mean proportion of correct answers as well (85.7% correct, *LessSkill*: 37.9% correct, OR = 9.4, 95CI = [2.6, 41], $d = 1.24$, $p < 0.001$).

For the time to correctly debug the tasks irrespective of treatment, *LessSkill* spent 73.1% more time than *MoreSkill* on correctly solving the Find task, $t(40) = 5.05$, $p < 0.001$, $d = 1.38$. Further, this group also spent 36.8% more time on correctly solving the Copy task, $t(21) = 2.32$, $p = 0.030$, $d = 0.80$, than *MoreSkill*. For Find, the mean time for *LessSkill* was 7.21 minutes (SD = 2.59, $n = 25$) and 4.16 minutes (SD = 1.75, $n = 32$) for *MoreSkill*. For Copy, the mean time for *LessSkill* was 7.49 minutes (SD = 2.69, $n = 13$) whereas *MoreSkill* used 5.47 minutes (SD = 2.36, $n = 28$). Moreover, when the results for Find and Copy were combined, *MoreSkill* was faster in correctly debugging as well ($p < 0.001$).

Hence, we can regard measures of skill as a relevant predictor of the debugging performance in the replication that may be used in the further analysis.

TABLE III. DEPENDENT VARIABLE CORRELATIONS WITH SKILL IN THE REPLICATION

Task	*Correctness*[a] *(n)*	*Time (n)*
Find	0.51 (64)	-0.56 (57)
Copy	0.55 (64)	-0.44 (41)

[a] point-biserial correlation

B. *Random assignment in the replication*

An implicit assumption in randomized designs is that randomization limits systematic effects of unequal groups with respect to factors that can significantly influence the experiment outcome [29, 40]. Although this holds true for large samples, when sample size is small it may pose a serious threat to the validity of inferences.

The subjects who were assigned to the recursive Find task had slightly higher mean skills than those who were assigned to the iterative version. The difference in mean skill, as estimated by the Rasch model, was 0.35 logits (OR = 1.42, $d = 0.19$). For the Copy tasks, the effect of randomized assignment to treatment was reversed; the iterative group had slightly higher skills than the recursive group on average ($\Delta = 0.13$ logits, OR = 1.14, $d = 0.07$). We now turn to the more detailed analysis where the effect of recursive and iterative treatments for the two tasks can be analyzed conditional on skill.

C. *Results*

In this section, we use Rasch analysis to expand the previously reported results. Information about each individual's skill is now used to estimate the difficulty of four *treatment pairs*: recursive Find, iterative Find, recursive Copy and iterative Copy. The procedure uses the principles of differential item functioning (DIF) that is commonly is used to investigate whether questions in a psychological test are biased towards subgroups such as non-native speakers of a language or ethnic minorities [see 17]. However, we perform some adaptations to traditional DIF analysis along the lines discussed in [23].

We will use the term *item* to denote one of the four treatment-task pairs above. The estimation process uses a conditional maximum likelihood function where residual (unexplained) variance is minimized. Further, unlike the differences in *mean* skill we reported in Section 4.2, the Rasch model accounts for skill differences on a *person-by-person* level. This implies that even if a group of individuals is more skilled on average, individual response patterns that yield more relevant information in determining the difficulty of tasks are used. Nevertheless, the inequality of treatment groups with respect to skill (as reported in Section 4.2) is now taken into consideration in determining the difficulty estimates for the four items.

All the four items were scored identically using three score categories: *Incorrect* = 0, *correct and slow* = 1, and *correct and fast* = 2. This score structure is a monotonic function of what "high performance" implies and uses an ordinal scale [15]. We (operationally) defined six minutes as the difference between "slow" and "fast" solutions, thereby roughly splitting the observations of Figure 1 in half. Although this procedure degrades time into a dichotomous variable, it allows time and correctness to be co-located on the same scale for more detailed analysis. This, in turn, makes it possible to express the difficulty of all four items as a function of skill.

Each individual's aggregated debugging score over both tasks (i.e., a sum score from 0–4) appeared normally

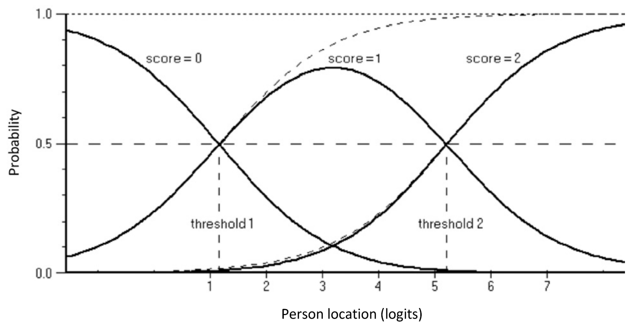

Figure 2. Category probability curves for different scores as a function of skill for the recursive Find task.

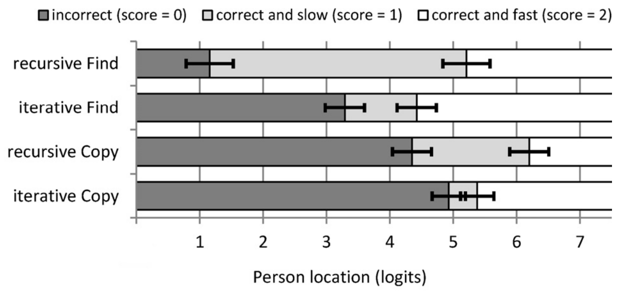

Figure 3. Estimated task difficulty thresholds by the Rasch model (threshold map).

distributed according to the Kolmogorov-Smirnov statistic (p = 0.058). Further, this sum score was also well predicted by skill (rho = 0.673, 95CI[1] = [0.484, 0.806], p < 0.001).

The Rasch model places item difficulty and a person's ability on the same interval scale [3]. An interval scale implies that additive transformations are permitted but not multiplicative transformations. Further, ratio interpretations are not meaningful because the number zero is not defined [45].

Figure 2 shows the expected probabilities for the recursive Find item using the three score categories (0–2) above. The mean population skill is transformed to be located at 5 logits and has a standard deviation of 1.3 logits (not shown) on the x-axis. Starting from the left, the figure shows that the incorrect (score = 0) response category has the highest probability for individuals with less skill than about 1.2 logits. Further, the probability of an incorrect solution decreases as skill increases, and the probability of an incorrect response becomes negligible at about 5 logits and above. Between 1.2 and 5.2 logits, the most probable response category is 1 (correct and slow). Above 5.2 logits, the most probable response category is 2 (correct and fast). The sum of the probability for the three score categories always equals 1.0 for any level of skill.

The dotted line that departs upwards from the "score = 1" category shows the cumulative distribution function for this score category. Because a score of 2 also implies that the first threshold has been passed [see, e.g., 4, 15], the probability of *not* achieving at least a correct and slow solution when skill is above about 5 logits is also negligible.

A *threshold* is the location on the logit scale where one response category replaces another one as being the most likely response. This is indicated in Figure 2 by two vertically dotted lines. There, the number of thresholds for each item equals the number of score categories minus 1: The *first threshold* is where score 0 and 1 intersect, and the *second threshold* is where score 1 and 2 intersect.

Figure 3 shows the most likely response category for all four items as a function skill. Although the exact category probability curves for each item is not shown (as in Figure 2), the differences in the location of thresholds provide the needed information about differences in difficulty between the four items. (The two thresholds in Figure 2 can be found for "recursive Find" row in Figure 3.) The standard error of measurement for each threshold is represented by a horizontal bar. Overall, the lower difficulty of the first threshold for both recursive tasks compared with their iterative alternatives supports the findings of original study: the recursive versions of the two tasks are easier to debug correctly. The effect is much larger for the Find task (Δ 2.1 logits) than for the Copy task (Δ 0.6 logits).

The second threshold represents the difficulty of debugging an item correctly in a "slow" versus "fast" manner. As expected, it is more difficult to achieve a correct and fast solution than a correct and slow solution. The results for the second thresholds were reversed with respect to what the better treatment was for both tasks; the iterative versions were easier to debug correctly and fast than the recursive versions of the tasks. However, inspecting the width of the standard errors in Figure 3 shows that none of the differences in threshold locations are significant; a 95% confidence interval for item thresholds can be obtained by roughly doubling the width of each standard error bar.

Based on the information contained in the threshold map in Figure 3, it is now possible to turn to a concrete example of how "what is the better treatment" varies as a function of skill. We first define three (hypothetical) groups: the *low-skilled* group has a skill of 3 logits (at 7th percentile), the *average-skilled* group has the mean skill of the investigated population and the *high-skill*ed group has a skill of 7 logits (at 93rd percentile). (The low- and high-skilled groups are about ±1.5 standard deviations below or above the mean skill.)

In Figure 4 we have combined the probabilities for the Find and Copy tasks. For the low-skilled group, the recursive implementations appear best because the probability of incorrectly debugging these tasks (0.45) is lower than for the iterative versions (0.69). At the same time, the difference in probability between the treatments for a correct and fast implementation also appears negligible (0.05 for recursive versus 0.06 for iterative).

For the high-skilled group, the iterative versions appear best. The expected probability of a correct and fast implementation for the iterative versions of the tasks combined is 0.88, whereas the corresponding probability for

[1] Calculated using PASW™ 18.0 for 2500 bootstrapped samples.

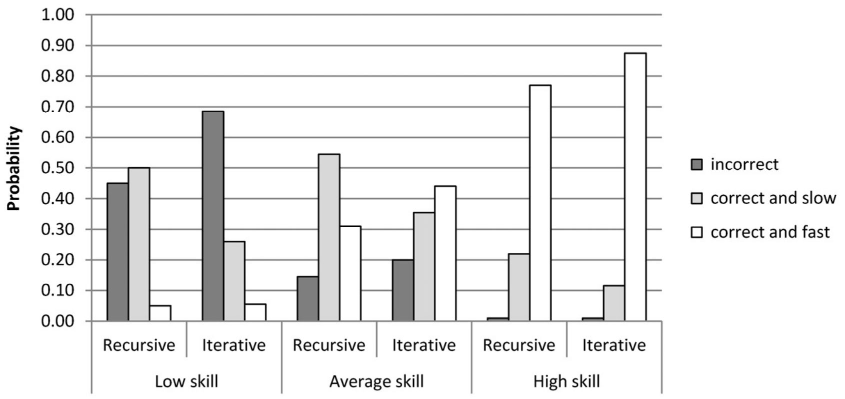

Figure 4. Expected score category probabilities for Find and Copy combined depending on skill and treatment.

the recursive versions is 0.77. At the same time, the probability for an incorrect solution for both recursion and iteration is only 0.01.

For the average-skilled group, the results are inconclusive, because a choice must be made with respect to preference in a time-quality tradeoff [see, e.g., 26]. For example, the iterative versions have a slightly higher probability of being incorrect (0.20 versus 0.15), whereas the probability of a correct and fast solution is also slightly higher (0.44 versus 0.33). Whether recursion or iteration is the better treatment for this group cannot be decided because there are no negligible differences for any of the score categories.

V. DISCUSSION

This section discusses and contrasts the results from the original and replicated study, first with respect to implication for research and then with respect to implications for practice. We then address limitations of our replicated study and discuss issues for future work.

A. *Implications for research*

In this study, the largest effect on debugging performance was neither the treatment nor the task complexity; it was the skill of the subjects. Overall, the two recursive implementations were slightly easier to debug correctly (OR = 1.50) than the iterative implementations. The original study had a similar result, although the effect size was smaller (OR = 1.21). By pooling the data from both studies (Table II), the effect of recursion being easier to debug correctly is marginal (OR = 1.18, 95CI = [0.85, 1.63], $d = 0.09$, $p = 0.34$).

Both studies show that the difference in difficulty between the tasks is larger than the effect of treatment. Combining the studies, the standardized effect size of task difficulty irrespective of treatment is between small and medium (OR = 1.87, 95CI = [1.34, 2.6], $d = 0.35$, $p < 0.001$).

However, in our replication, these effect sizes are dominated by the effect of individual differences in skill: When correctness for Find and Copy was merged and analyzed irrespective of treatment, the more skilled group had 92.9% correct solutions and the less skilled group had 56.9% correct solutions. This difference represents a large standardized effect size (OR = 9.4, 95CI = [2.6, 41], $d = 1.25$, $p < 0.001$) that ranks in the top 25th percentile of 284 software engineering experiments [see 28]. Differences in skill must therefore be controlled for in empirical studies of programmers.

Generalization over tasks usually requires that the results be consistent across several tasks. The original study had only two tasks. Consider Segal's law: "a man with a watch knows what time it is. A man with two watches is never sure." Only by having multiple operationalizations is it possible to make more qualified inferences on the extent to which a result can be generalized. When expressing the effect of treatment as a function of skill by using the Rasch model, we obtained results that were consistent across two tasks despite the challenge that a large task difficulty factor presented.

The combined results of three studies now support the conclusion of the authors of the original study. Table IV shows the overall results for the comprehension, original, and replicated studies, where "+" denotes positive support of the original authors' conclusion with respect to debugging correctness. Yet, their large study still failed to yield results in support of the recursive version of the Find task as being easier to debug. Although the difference of the effect size of recursion versus iteration for the two tasks is relatively small and difficult to detect, the sample size of the original study requires us to conjecture why they did not find that the recursive Find task was easier than the iterative one.

A *practice effect* is when performance increases for each

TABLE IV. SUPPORT FOR RECURSION BEING MORE EASY TO DEBUG CORRECTLY

Study	***# responses (both tasks)***	***Debugging phase***	***Task***	
			Find	***Copy***
Comprehension [10]	275	Identification	+	(–)
Original [11]	531		+	+
		Correction	(–)	+
This replication	128		(+)	(+)

+ denote positive support and – denote negative support for the original authors' conclusion
( ) represents non-significant results

new task in a study when the performance is supposed to be stable in order not to bias the study [41]. Practice effects are common threats to validity in within-subject designs [40] and are known to increase individual variability and thereby decrease the statistical power to detect differences. In the skill instrument, we have previously reported the presence of a small "warm-up" effect for the first three tasks (1–2 hours), but it is not present afterwards [14]. It is therefore tenable that the original study, which only involved two small tasks, is influenced by a similar practice effect (any potential practice effect in our replication is averaged over 19 tasks using randomization).

The mean degree of correctness for Find and Copy for the students in the original study and the professionals in the replicated study (Table II) deserves to be addressed in order to address a potential practice effect. The student's probability of correct solutions for their first (Find) task was 0.53 lower than that for the professionals, but only a difference of 0.40 separated the two populations for the second (Copy) task. This indicates that the students improved their performance on their second task more than did the professionals. Typically, professionals are more skilled than students and therefore learn less from practice [see 26]. Nevertheless, a systematic improvement in performance during a study is problematic, because it implies that the subjects are not well versed in using the technology; hence the results cannot be generalized [41]. An improvement of 0.13 (i.e., 0.53-0.40) in the proportion of correct responses is almost equal in size to the difference in mean difficulty of the two tasks (0.16). Hence, it may appear that that a practice effect, in addition to the effect of skill and task difficulty, may also be larger than the effect of treatment in the original study.

B. *Implications for practice*

We found weak results in favor of iteration being easier to debug fast and correctly than recursion. Although this result was not significant, it was consistent over both tasks.

It is self-evidently true that a technology is better when it is easier *and* faster to use than when it is not. However, what if a "faster" technology comes at the price of added complexity, which makes the technology harder to use properly? Then the faster technology would require more training to be used successfully. Without training, the faster and more complex technology would be associated with a higher proportion of incorrect uses, thereby making the faster technology appear worse than the existing alternative that is slow but easy to use correctly already.

There are several examples of occasions when a faster technology is more difficult to use. For example, a bicycle is a faster means of transportation than walking, but one must know how to ride it. Similarly, a typewriter is easier to use than a computer, but the computer is faster for text editing when used correctly. More complex technologies frequently require more training than less complex technologies; at the same time, more complex technologies are adopted because they add to productivity.

Fundamental to our reported results that the potential advantage of different debugging implementations may depend on skill levels are two basic assumptions. First, a correct solution is better than an incorrect solution. Second, a fast solution is better than a slow solution *if* both solutions are correct [21]. As previously shown in Fig 4., when the probabilities for an incorrect solution is high, it is normal to take steps to improve the degree of correctness before less time becomes an important factor. This seems to be the case for our (hypothetically defined) low-skilled subjects. An opposite situation was present for the high-skilled subjects: because the proportion of incorrect and correct answers was negligible whereas the difficulty of a fast and correct solution was lower for the iterative versions, the preference for what treatment was better was reversed. The tradeoff between quality and time is certainly present when practitioners evaluate the benefits of new software engineering technologies.

C. *Limitations*

A limitation of this replication is low statistical power. Although we had sufficient power to detect large and systematic differences in the skills of the subjects and the differences in the difficulty of the tasks, our results with respect to the treatments were not significant. Further, even though the study magnitude of our replication is large according to the conventions of [43] (the professionals spent more than 1000 hours combined), the Rasch model can be data intensive when the purpose is to characterize individual differences [see 31]. Because our research question considers *group* differences (rather than individual differences), fewer than the recommended number of subjects for using the Rasch model is therefore acceptable. We also regard the limitation of only having two debugging tasks to generalize from as a greater concern than the statistical power at present.

In this replication, there was only one response for the "incorrect" score category for the recursive Find task. This implies that the standard error associated with the first threshold for this task is not adequately represented. Ideally, all response categories should be well populated to obtain accurate item thresholds in the Rasch model. For the two tasks investigated here, a new sample of less skilled subjects is needed to obtain lower standard errors of measurement in the item difficulty thresholds.

D. *Future work*

To measure the skill of the subjects in this study we used a specifically tailored research prototype. We are now working on making the skill instrument industry strength. New or replicated experiments may then be administered as part of ongoing assessments of professionals and students, something that facilitates the use of more statistically powerful experimental designs (e.g., matching or pairing, see [40]). Such designs are particularly relevant for studies where few subjects are available, within-subject designs are not feasible, or where random assignment to treatment is not possible. We will also conduct more studies where skill is taken into account when investigating the effect of a technology. We welcome future collaborations.

VI. CONCLUSION

An implicit assumption in many research studies in software engineering is that the benefit of a new technology or method is invariant of skill levels. The study reported in this paper illustrates why such an assumption is problematic. Using a measurement model where the effect of recursive versus iterative implementations of two small debugging tasks was expressed as a function of skill, we provided additional evidence that "what is the better" of two competing technologies requires the additional qualifier "for whom?"

We found that for the low-skilled subjects, the results were in favor of recursive implementations, which supports the original study. An opposite result was found for the high-skilled subjects; the iterative versions were debugged faster while the difference in the proportion of correct answers compared to the recursive version was negligible. Hence, the benefit of debugging the iterative versions (less difficult to debug fast but more difficult to debug correctly), is based on an important principle: The probability of incorrectly using both debugging alternatives must be low and negligible before the faster technology can be assumed to be better.

This study does not stand in isolation; previous large-scale experiments have reported similar interaction effects between skill levels and the technology or method being investigated. Still, there is often a gap between researcher expectations and empirical results because one fails to acknowledge that potentially more powerful technologies may be more complex to use, or may require new skills in order to use correctly. The community must raise its awareness of how skill levels, which in this study was much larger than the difference between treatments, affect the claimed benefits of alternatives being evaluated. Consequently, we need better ways to measure relevant skills with respect to the product or process being investigated.

ACKNOWLEDGMENT

This research was supported by the Research Council of Norway through the FORNY program, and by Simula Research Laboratory. We thank Steinar Haugen and Gunnar Carelius for technical assistance, and Magne Jørgensen for reading through an earlier version of this manuscript. We also thank Christian Brinch for pointing out that our pair-wise comparison can be analyzed using a binominal test.

REFERENCES

[1] B. C. D. Anda, D. I. K. Sjøberg, and A. Mockus, "Variability and reproducibility in software engineering: A study of four companies that developed the same system," IEEE T. Software Eng., vol. 37, pp. 407–420, 2009.

[2] J. R. Anderson, "Skill acquisition:Compilation of weak-method problem solutions," Psychol. Rev., vol. 94, pp. 192–210, 1987.

[3] D. Andrich, "A rating formulation for ordered response categories," Psychometrika, vol. 43, pp. 561–573, 1978.

[4] D. Andrich, Rasch models for measurement, CA: Sage Publications, 1988.

[5] D. Andrich, B. Sheridan, and G. Luo, RUMM2020 [computer software]. Perth: RUMM Laboratory, 2006.

[6] 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, pp. 65–86, 2007.

[7] 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, pp. 521–534, 2004.

[8] E. Arisholm, D. I. K. Sjøberg, G. J. Carelius, and Y. Lindsjørn, "A web-based support environment for software engineering experiments," Nordic J. Comput., vol. 9, pp. 231–247, 2002.

[9] V. C. Basili, F. Shull, and F. Lanubile, "Building knowledge through families of experiments," IEEE T. Software Eng., vol. 25, pp. 456–473, 1999.

[10] A. C. Benander, B. A. Benander, and H. Pu, "Recursion vs. iteration: An empirical study of comprehension," J. Syst. Software, vol. 32, pp. 73–83, 1996.

[11] A. C. Benander, B. A. Benander, and J. Sang, "An empirical analysis of debugging performance—differences between iterative and recursive constructs," J. Syst. Software, vol. 32, pp. 73–83, 1996.

[12] 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.

[13] 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., vol. 32, pp. 201–209, 2011.

[14] G. R. Bergersen and J. E. Hannay, "Detecting learning and fatigue effects by inspection of person-item residuals," 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/dqiu2.

[15] G. R. Bergersen, J. E. Hannay, D. I. K. Sjøberg, T. Dybå, and A. Karahasanović, "Inferring skill from tests of programming performance: Combining time and quality," Proc. International Symposium on Empirical Software Engineering and Measurement (ESEM), IEEE Computer Society, Sep. 2011, pp. 305–314.

[16] T. G. Bond and C. M. Fox, Applying the Rasch model: Fundamental measurement in the human sciences. Mahwah, NJ: Erlbaum, 2001.

[17] D. Borsboom, "When does measurement invariance matter?" Med. Care, vol. 44, pp. S176–S181, 2006.

[18] R. E. Brooks, "Studying programmer behavior experimentally: The problems of proper methodology," Commun. ACM, vol. 23, pp. 207–213, 1980.

[19] F. P. Brooks, The Mythical Man-Month: Essays on Software Engineering, Anniversary ed. Reading, MA: Addison-Wesley, 1995.

[20] A. Brooks, J. Daly, J. Miller, M. Roper, and M. Wood, Replication of experimental results in software engineering. University of Strathclyde, International Software Engineering Research Network (ISERN) Technical Report, 10, 1996.

[21] J. B. Carroll, Human cognitive abilities. Cambridge: Cambridge University Press, 1993.

[22] R. B. Cattell, Abilities: Their structure, growth, and action. Boston: Houghton-Mifflin, 1971/1987.

[23] W.-C. Chang and C. Chan, "Rasch analysis for outcome measures: Some methodological considerations," Arch. Phys. Med. Rehabil., vol. 76, pp. 934–939, 1995.

[24] S. Chinn, "A simple method for converting an odds ratio to effect size for use in meta-analysis," Statist. Med., vol. 19, pp. 3127–3131, 2000.

[25] J. Cohen, "A power primer," Psychol. Bull., vol. 112, pp. 155–159, 1992.

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

[27] R. R. Glass, "Frequently forgotten fundamental facts about software engineering," IEEE Software, pp. 110–112, May/June 2001.

[28] V. B. Kampenes, T. Dybå, J. E. Hannay, and D. I. K. Sjøberg, "A systematic review of effect size in software engineering experiments," Inform. Software Tech., vol. 49, pp. 1073–1086, 2007.

[29] 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, pp. 71–82, 2009.

[30] J. J. Krein, C. D. Knutson, L. Prechelt, N. Juristo, "Report from the 2nd International Workshop on Replication in Empirical Software Engineering Research (RESER 2011)," SIGSOFT Software Engineering Notes, vol. 37, pp. 27–30, 2012.

[31] J. M. Linacre, "Sample size and item calibration stability," Rasch Measurement Transactions, vol. 7:4, pp. 328, 1994.

[32] S. E. Maxwell, "Covariate imbalance and conditional size: Dependence on model-based adjustments," Stat. Med., vol. 12, pp. 101–109, 1993.

[33] J. C. Nunnally and I. H. Bernstein, Psychometric theory, 3rd ed. NY: McGraw-Hill, 1994.

[34] R. Ostini and M. L. Nering, Polytomous item response theory models. CA: Sage Publications, 2006.

[35] L. Prechelt, "Plat_Forms: A web development platform comparison by an exploratory experiment searching for emergent platform properties," IEEE T. Software Eng., vol. 37, pp. 95–108, 2011.

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

[37] R Development Core Team (2008), R: A language and environment for statistical computing [computer software v. 2.13.2]. R Foundation for Statistical Computing, 2008. http://www.R-project.org.

[38] G. Rasch, Probabilistic models for some intelligence and attainment tests. Copenhagen: Danish Institute for Educational Research, 1960.

[39] F. L. Schmitt 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, pp. 262–274, 1988.

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

[41] B. A. Sheil, "The psychological study of programming," ACM Comput. Surv., vol. 13, pp. 101–120, 1981.

[42] D. I. K. Sjøberg, B. Anda, E. Arisholm, T. Dybå, M. Jørgensen, A. Karahasanovic, E. Koren, and M. Vokác, "Conducting realistic experiments in software engineering," Proc. First International Symposium on Empirical Software Engineering (ISESE), IEEE Computer Society, Oct. 2002, pp. 17–26.

[43] D. I. K. Sjøberg, J. E. Hannay, O. Hansen, V. B. Kampenes, A. Karahasanovic, N.-K. Liborg, and A. C. Rekdal, "A survey of controlled experiments in software engineering," IEEE T. Software Eng., vol. 31, pp. 733–753, 2005.

[44] V. J. Shute, "Who is likely to acquire programming skills?" J. Educ. Comput. Res., vol. 7, pp. 1–24, 1991.

[45] S. S. Stevens, "On the theory of scales of measurement," Science, vol. 103, pp. 677–680, 1946.

[46] L. L. Thurstone, "Attitudes can be measured," Am. J. Sociol., vol. 4, pp. 529–554, 1928.

[47] D. J. Woltz, "An investigation of the role of working memory in procedural skill acquisition," J. Exp. Psychol. Gen., vol. 117, pp. 319–331, 1988.