This is a preprint version of: Bergersen, G. R. and Sjøberg, D. I. K. (2012). Evaluating methods and technologies in software engineering with respect to developers' skill level. In 16th International Symposium on Evaluation & Assessment in Software Engineering(EASE'2012) (pp. 101-110). Ciudad Real, Spain: IET.

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 bio-graphical 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

Accepted 2012-Mar-04 Not suitable for citation Copyright IET 2012

This is a preprint version of: Bergersen, G. R. and Sjøberg, D. I. K. (2012). Evaluating methods and technologies in software engineering with respect to developers' skill level. In 16th International Symposium on Evaluation & Assessment in Software Engineering(EASE'2012) (pp. 101-110). Ciudad Real, Spain: IET.

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

 Copyright IET 2012

This is a preprint version of: Bergersen, G. R. and Sjøberg, D. I. K. (2012). Evaluating methods and technologies in software engineering with respect to developers' skill level. In 16th International Symposium on Evaluation & Assessment in Software Engineering(EASE'2012) (pp. 101-110). Ciudad Real, Spain: IET.

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.

 Copyright IET 2012