Ralf Teusner, Thomas Hille: On the Impact of Programming Exercise Descriptions, In Proceedings of 2018 Learning With MOOCS (LWMOOCS2018), 26-28 Sept. 2018, Madrid, Spain,

DOI: 10.1109/LWMOOCS.2018.8534676,
Link to published version: https://ieeexplore.ieee.org/document/8534676

On the Impact of Programming Exercise Descriptions

Effects of Programming Exercise Descriptions to Scores and Working Times

Ralf Teusner
Hasso Plattner Institute
University of Potsdam
Potsdam, Germany
ralf.teusner@hpi.de

Thomas Hille
Hasso Plattner Institute
University of Potsdam
Potsdam, Germany
thomas.hille@student.hpi.de

***Abstract*—This paper examines the effects of exercise descriptions and supplied hints to achieved scores and required working times of programming exercises within a MOOC. We conducted an A/B test on more than 2.400 students using an exercise with four descriptions differing in clarity, description detail, order of presented instructions and presence or absence of exemplary program output. Results show that the expressiveness and structure of the instructions influence the required work times as well as the number of issued program runs. We suggest further experimentation to outline effects on programming errors and to determine guidelines for optimal exercise descriptions.**

Keywords—programming, exercise, description, working time

I. Introduction

Recalling school, when asked which exercises students perceived as the most difficult ones, the typical answer is: word and real life problems. They require students not just to calculate results or apply given instructions, but demand understanding of a problem, combination of the given facts, determining a suitable approach and finally solving the problem. The hardest parts are usually: understanding the problem and finding a suitable approach. When offering practical programming exercises within MOOCs, the requirements are similar: students have to understand the problem, build on the given code snippets, apply the explained concepts and code the solution. Therefore, the exercise descriptions play a vital role in the success of an offered exercise. In this paper, we analyze the effect of different exercise descriptions, each offering different levels of guidance and hints, on the same practical programming exercise with regards to exercise completion rates, scores and the time required.

II. Related Work

The effect of structure, wording, and topic of so-called word or real world problems is mostly evaluated on math problems and in school classes. Palm investigated the impact of authenticity in a 5th graders math class and found improved reasoning and more detailed answers were effects on more authentic exercises [1]. Nortvedt, Gustafsson, and Lehre found a strong correlation between reading and mathematics in a study on grade four students in 34 countries. However, the effect of the instructional quality of teachers could not be evaluated towards a general conclusion [2]. Vicente and Manchado also analyzed the effects of authentic wording in math problems and concluded that authentic problems improve the results especially for high skilled students [3]. Heidelberger evaluated the impact of word problems on high school students. However, his studies were mostly focused on the task of creating own word problems and therefore training the opposite direction of typical word problems. It improved their understanding and transferring from problem description from to mathematical models and vice versa [4].

Our experiment is not covering the entire range of a real world problem, requiring students to map a complete situation or story to program code. We only require our students to transfer the more or less detailed information of an already mostly mapped exercise description to program code. This experiment is therefore also impacted by research conducted in the field of instructional scaffolding and hinting, as we offer different levels of details for our problem, by including or excluding examples and hints and formulating the question on different levels of clarity. Scaffolding approaches are often used for learning programming in general, but literature lacks examples of the effects of. Ismail, Ngah, and Umar identify and describe the main problems of novice programmers in general, such as a lack of skills to analyze and represent abstract issues [5]. However, they do not go much further than stating the problem. Caspersen and Bennedsen also identify these problems, but approach them from the perspective of different learning theories and propose a comprehensive course structure to tackle these problems [6]. Also, Lindner, Abbott, and Fromber designed a full series of tasks to teach their students software design using a scaffolding approach [7]. Thus, existing literature mostly approaches the macro level of course design, but lacks detail on the micro level, such as specific exercise construction. While these findings imply some hypotheses on the macro level, research is sparse for computer science problems on a micro level and nearly non-existent for the specific field of programming tasks in MOOCs. With this work, we therefore contribute some first experimental findings to a vast but currently open field to give some general directions for future research.

III. Practical Programming Exercises

A. Implementation and Grading

Our programming exercises are automatically graded on our code execution platform CodeOcean[1] and seamlessly

[1] https://github.com/openHPI/codeocean

integrate into the MOOC through LTI[2]. Unit tests are executed on students' requests. In addition to scores and completion rates, the coding platform implicitly gathers approximate working times as students progress through the exercises by submitting time-stamped solutions.

B. Characteristics

Our programming exercises are a vital part of the course and are interleaved with video lectures. For every new concept, for example "method calls", we start with a video lecture to impart theoretical knowledge. Afterwards, we establish and fortify practical knowledge with hands-on exercises: the first exercise asks to re-implement the exact steps shown in the video, the second exercise usually alters some parameters, and the third exercise challenges the student to apply the concept in a suitable but different situation, to train students to discover applicable use cases themselves. At the end of each course week, we occasionally offer exercises that require students to revisit and combine different taught concepts.

IV. Concept And Study Design

The experimental task discussed in this paper can be solved within one method of one class in Java and was integrated into a German MOOC with 9.242 registered students, 5.839 of them actively participating. We split our students into four evenly distributed groups based on their artificial user id and offered all of them the additional exercise as an optional bonus task during the second course week. The exercise asks students to use a for-loop to print out 50 lines and to adjust the output depending on the current loop iteration[3]. If the counter is divisible by 3, it should print "ding"; if it is divisible by 7, it should print "dong"; if it is divisible by 3 as well as 7, it should print "ding-dong". In all other cases it should just print "ping". The exercise requires the student to have understood for-loops, as well as if-conditions, divisions and the modulo operator. All prerequisites were explained and fortified with distinct exercises beforehand. Depending on the experiment group, we gave the students more or less direct instructions and alternated the order in which we presented the conditions necessary to consider to solve the task correctly. The differences between the four experiment groups a) to d) are summarized below.

a) Instructions were given in optimal order, so first check whether dividable by 3 and 7, then only by 3 or 7, then the else case. This resembles the order the conditions have to be placed in the if-conditions. We further gave the students the hint that the divisibility without remainder can be checked with the modulo operator. These students were given the first 10 lines of expected output.

b) Instructions were in incorrect order, first checking by 3, then by 7, then by 3 and 7, then the else case. The else case was presented more complicated then necessary, by saying "if it is neither divisable by 3 or 7, …". The students were also given the first 10 lines of expected output.

c) We gave the instructions in incorrect order, and added unnecessary information. We first asked them to check whether the counter can be divided by 3, but not by 7, then to check whether its dividable by 7, but not by 3, and then check whether the counter can be divided by both. In all other cases, the ping output should be presented. The students were not given any expected output, but we gave them the hint that they should think about the order in which the checks should be done, before implementing it.

d) The shortest description just stated: "for every iteration dividable by 3, print out 'ding', for every iteration dividable by 7 'dong', and for iterations that are dividable by 3 as well as 7 print 'ding-dong'. If an iteration is neither dividable by 3 nor by 7, print 'ping'." The description did not contain any expected output or further hints.

V. Evaluation

A. Results

2,444 students accessed the exercises, most of them finishing within less than 15 minutes. The starting numbers were necessarily similar around 650, as we distributed the students evenly. The completion rates also did not differ significantly; they ranged between 79% and 83%. For the following analysis, we sanitized the data by only considering solutions that took between 1 minute and 1 hour to complete, thus removing extreme outliers. Results are shown in Table 1. The working times and program runs moderately correlate for the four exercises under consideration (Pearson correlation coefficient = 0.68, $p = 0.3$). For just four items, there is no statistical significance. However, over all exercises of the complete course, the working times and program runs have a strong correlation of 0.99 ($p < 0.005$). The standard deviations of all measurements are relatively high, especially for the working times and average number of runs, ranging between 76% and 94%.

TABLE I. Core Metrics of the Experiment Exercises, Standard Deviations in Percentage of the Mean Metric

Exer-cise	∅ score, rel. stdev	∅working time [min], rel. stdev	∅ runs, rel. stdev
A	0.973, 29%	11:40 (base), 79%	7.1 (base), 85%
B	0.976, 29%	12:58 (+11%), 78%	6.8 (- 4%), 83%
C	0.974, 36%	13:10 (+13%), 76%	7.5 (+ 6%), 74%
D	0.967, 26%	13:09 (+13%), 82%	8.8 (+24%), 94%

With regards to statistical tests, we conducted Welch two sample t-tests on the mean working times. Each test compared the more complicated exercise versions B, C, and D with the baseline exercise A. The test supports that for the combination of working times of exercises A and C, we can reject the null hypothesis that the true difference in means is equal to 0 (and thereby the two samples are likely to originate populations with the same mean), with significance as it resulted in $p < 0.005$. For combinations A and B ($p = 0.02$), as well as A and D ($p = 0.01$), we can't conclude this with statistical significance.

[2] Learning Tools Interopability, see https://www.imsglobal.org/activity/learning-tools-interoperability

[3] A variation of the so called FizzBuzz exercise, used to train division.

We further gathered additional feedback by surveying 10% of our participants after their final submission. When asked for difficulty ratings, the impressions did not differ between the exercises and ranged between simple and average difficulty. Free text answers repeatedly mentioned that they liked the exercise as it combined several learned constructs and thus repeated the learned concepts in a suited context. Statistical significance of these answers is not given, as the number of surveyed participants was too low.

B. Discussion

As with all our exercises, average scores are of limited value. Students show high diligence, so our measurements show a ceiling effect and there is not much room for improvement. Average working times and the mean number of program runs better reflect students' difficulties as well as general progress. The high standard deviations reflect large differences in individual students skills. Some students finished their exercises within slightly more one minute, others needed nearly an hour, and several outliers of each group needed almost two hours. The outliers were however also uniformly distributed. The explanations of exercises A and B are very descriptive and showed the output of the required program. The descriptions of exercises C and D are much shorter and less straightforward, without showing the expected output. In contrast to D, exercise C includes a hint. The results show that longer, more helpful descriptions may need longer to read, but in the end, cause shorter average working times. We also notice that more complicated texts increase the difficulty for students to solve an exercise, reflected by an increase in average working time and even more prominent, the number of exercise submissions (runs). This is also further supported by the minor decrease in average score. If exercise A is taken as a baseline, as it had the most helpful exercise description, the more difficult exercise descriptions caused an increase of up to 13% in working time, while students furthermore needed up to 24% more program runs on average. The high increase in program runs for exercise D is likely caused by the rather minimal description, thereby requiring the students to conduct more trial runs to gauge desired behavior and more checks against the supplied tests.

The t-tests support that exercise C resulted in significantly different behavior with regards to the average working times compared to our baseline exercise A. We therefore conclude that the quality of the exercise descriptions has a considerable effect towards the students' experiences. Although every exercise description contains all required information, a less guiding description increases the actual difficulty, reflected by the required completion time. In our case, the perceived difficulty, reflected by students' responses in the survey, did not differ. Our assumption is that either the increase in difficulty was not high enough to be noticed, or the students determine the perceived difficulty on the basis of the general problem category of a FizzBuzz exercise. When reaching the survey, the participants already solved the exercise, thus had understood the problem and therefore were no longer dependent on the exercise description.

Comparing our findings with those in related work, we argue that the results are consistent with the ones brought up by Nortvedt, Gustafsson, and Lehre. Artificially impeding initial problem comprehenshion by presenting a harder text affects students' performance. Our expectation is that existing findings from classic education literture are in general transferrable to the domain of programming exercises, even though word problems and programming exercise descriptions differ in abstraction levels. This hypothesis is supported by our measurements.

VI. Future Work

Existing, faulty runs can be used in order to improve exercise descriptions, if common errors can be outlined. As of now, this is a manual, and therefore tedious as well as error-prone, task. In the future, we plan to automatically analyze and group occurred errors in order to determine whether specific descriptions cause certain errors to be more likely in a repeated experiment.

VII. Conclusion

When designing a MOOC or a learning resource in general, much effort is spent on the didactic concept, the order that learning resources are presented in, as well as the time spent on specific concepts. We explored the domain of practical programming exercises with regards to learning success. Working times and the number of trails against the solution are a valid and suitable approach to complement user surveys on self-stated success. Our results emphasize that educators should keep in mind that the results of programming exercises depend on the difficulty of the task, which is in term is significantly influenced by wording, structure and given hints. When investigating potential weaknesses in course material, these factors thus should be taken into serious consideration, together with the quality of the core instructional resources. The optimal balance between challenge, guidance, and repetition for programming content has to be found on an individual basis per topic. However, general insights on the minimum and maximum of text lengths and expressiveness of small, well defined training exercises seem to be detectable. We therefore strongly encourage future experimentation in this field.

References

[1] T. Palm, "Impact of authenticity on sense making in word problem solving" Educational Studies in Mathematics, vol. 67, 2008, pp. 37-58.

[2] G. A. Nortvedt, J. Gustafsson, and A. W. Lehre, "The Importance of Instructional Quality for the Relation Between Achievement in Reading and Mathematics.", 2016. Vol. 2. pp. 97-113.

[3] S. Vicente and Eva Manchado, "Arithmetic word problem solving. Are authentic word problems easier to solve than standard ones?" in Infancia y Aprendizaje, vol. 39, Routledge, 2016, pp. 349-379.

[4] J. Heidelberger, "Student-Authored Word Problems and Their Impact on High School Mathematics Students' Engagement" (2013). Masters of Arts in Education Action Research Papers. Paper 8.

[5] M. N Ismail, N. A. Ngah, and I. N. Umar, "Instructional Strategy in the Teaching of Computer Programming: A Need Assessment Analyses" The Turkish Online Journal of Educational Technology, vol. 9/2, 2010

[6] M. Caspersen and J. Bennedsen, "Instructional Design of a Programming Course – A Learning Theoretic Approach" Proceedings of the Third International Workshop on Computing Education Research, pp. 111-122, 2007

[7] S. P. Linder, D. Abbott, and M. Fromber, "An Instructional Scaffolding Approach to Teaching Software Design", ournal of Computing Sciences in Colleges, vol. 21, issue 6, pp. 238-250, 2006