Impact of Contextual Tips for Auto-Gradable Programming Exercises in MOOCs

Sebastian Serth
Hasso Plattner Institute
Potsdam, Germany
sebastian.serth@hpi.de

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

Christoph Meinel
Hasso Plattner Institute
Potsdam, Germany
christoph.meinel@hpi.de

ABSTRACT

Learners in Massive Open Online Courses offering practical programming exercises face additional challenges next to the actual course content. Beginners have to find approaches to deal with misconceptions and often struggle with the correct syntax while solving the exercises. The paper at hand presents insights from offering contextual tips in a web-based development environment used for practical programming exercises. We measured the effects of our approach in a Python course with 6,000 active students in a hidden A/B test and additionally used qualitative surveys. While a majority of learners valued the assistance, we were unable to show a direct impact on completion rates or average scores. We however noticed that users requesting tips took significantly longer and made more use of other assistance features of the platform than users in our control group. Insights from our study can be used to target beginners with more specific hints and provide additional, context-specific clues as part of the learning material.

Author Keywords

tip; hint; programming; summary; example; exercise; MOOC

CCS Concepts

•**Social and professional topics** → **Computer science education;** •**Applied computing** → **E-learning;** *Interactive learning environments;*

INTRODUCTION

Massive Open Online Courses (MOOCs) covering programming education are increasingly popular and provide a low barrier for beginners to start learning. However, most novices find programming difficult and, in distance education, spend multiple hours on try-and-error approaches without achieving the desired effect. While it is beneficial for beginners to find a solution themselves, clueless guessing does not allow learners to reflect on mistakes [2]. Since beginners often have problems summarizing their code-level questions meaningfully, several efforts have been made to support learners directly within the programming environment.

L@S'21, June 22–25, 2021, Virtual Event, Germany

ACM ISBN 978-1-4503-8215-1/21/06.
DOI: https://doi.org/10.1145/3430895.3460166

In this paper, we introduce and evaluate contextual tips provided by the teaching team for programming exercises as an additional effort to assist learners in their help-seeking behavior. We address the following research questions:

RQ1. How do tips influence the help-seeking behavior?

RQ2. Which learners profit more from tips than others?

RQ3. Do tips have an impact on key metrics such as the completion rate, working times, or scores?

RELATED WORK

This paper contributes to the field of Computer Science Education with MOOCs where (next-step) hints and help-seeking behavior has been of special interest recently.

Hints in Programming Education

Depending on the grading approach, some of the most prominent examples of contextual hints include tips for unit tests, incremental to-do notes, or annotations in the source code.

Providing scaffolded source code is proven to direct the learners' focus on specific aspects of an assignment [3] and can optionally include source code comments to ease navigation or highlight open to-dos. In a CS1 course, Antonucci et al. offered students the ability to get textual hints through to-do notes or to reveal parts of a solution. Their research indicates the need for a hint system in a beginner course and outlines the willingness of students to accept hints [1].

Other approaches to automatically generate hints are compared by Price et al., who focus on next-step hint generation [9]. These systems use submissions from various learners for a given problem to infer hints representing a common problem-solving strategy. Hints generated by this system are specifically tailored for the user's problem but vary in their quality and also depend on previously available data [9].

In contrast to next-step hints, some auto-graders scoring submissions enrich test results to increase comprehensibility. Király et al. argue that textual hints along with failed test results will enable students to fix common problems faster [6]. In 2019, Marwan et al. measured the impact of adding a textual explanation to next-step hints on "novices' programming performance" [7]. The authors showed that beginners greatly benefit from a written explanation and thus were better able to relate the hint to their problem. Similar approaches improve the understandability of occurring errors by translating cryptic error messages to more human-readable explanations [4].

In addition to MOOCs teaching the basics of programming, many providers offer interactive web tutorials with an online code editor and written step-by-step instructions. A particular disadvantage of these approaches is the missing explanation of why and in which situations a concept should be used [5].

Help-Seeking Behavior

Throughout a given programming assignment, students might use earlier course materials or external information. However, for novices, "one source of complexity is the availability of a wide variety of information sources" [8]. Hence, previous research focussed on supporting students without overwhelming them. Serth et al. suggested that the presentation of content can support students in revisiting previous content [10].

Another approach is to introduce *Requests for Comments* [12]. These allow learners to ask fellow learners questions about their implementation and automatically include relevant source code together with the run and test output. The researchers have shown the positive impact of *Requests for Comments* and validated that they support the help-seeking behavior.

CONCEPT

Beginners in a programming MOOC struggling with a given exercise should have direct access to dedicated, high-quality resources while implementing code. One of the solutions we propose is to offer instantly available tips serving as a reminder for programming concepts. In our vision, tips are related to the given exercise but are not directly influenced by the current implementation. Therefore, problem-solving strategies, which participants need to develop during a MOOC to continue programming autonomously, are still being practiced.

In our concept, each tip consists of four parts: (1) a describing title, (2) a short summary of the concept, (3) a self-contained example, and (4) subsequent tips if applicable. The title is the only information students can read without interacting with the tip preventing users from unintentionally spotting a solution. A tip may include nested tips allowing learners to look up further information. Nested tips can either be used to provide incremental hints on the same topic becoming more specific towards a possible solution or can be used to structure different aspects of one topic into smaller components.

As we consider tips to be an integral part of the learning material and thus the course experience designed by course instructors, we refrain from automatically generating hints. Instead, each tip is contextually tailored for the specific exercise.

IMPLEMENTATION

We implemented a configurable framework for editing and displaying tips within the web-based programming environment *CodeOcean* [11]. Learners have direct access to the tips while implementing a given programming assignment. The list of tips is separated from other elements of the page ensuring that tips are perceived as subtle rather than visually distracting.

Tips are edited independently from exercises and thus can be reused multiple times. For each exercise, instructors select tips and can optionally build an ordered tree structure using a drag-and-drop editor. This supports the teaching team in nesting tips to provide different levels of information for learners.

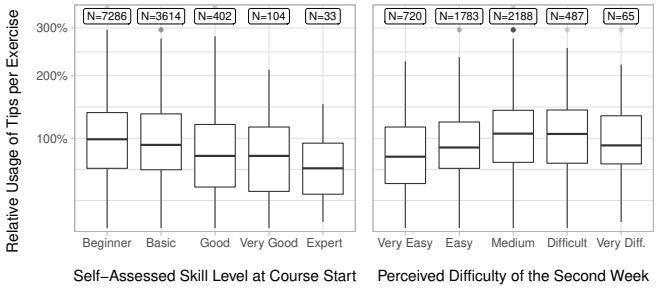

Figure 1. Relative number of tips used per exercise (left) according to the self-assessed skill level (N = 11,439), and (right) compared to the perceived difficulty (N = 5,243). A value of 100% represents the mean tips used by students per exercise. Black bars are median values.

EVALUATION

To validate our concept, we tested our implementation and gathered results from a field experiment.

Methodology

Targeting novices, we decided to conduct our evaluation in a four-week Python programming introduction MOOC. At the course end, 9,517 students were enrolled with a show rate of 79%. The teaching team manually authored tips for all exercises of the first two weeks. In the second week, the most basic hints were omitted in favor of tips covering new topics.

To measure the effects of the availability and usage of tips, we conducted an A/B testing within *CodeOcean*. Learners were randomly assigned with a probability of 50% to either the treatment group with tips or the control group without tips. Regular surveys were used to capture feedback about the tips.

Results

In total, 6,067 learners accessed at least one programming exercise in *CodeOcean* and thus participated in our A/B testing setup. 3,032 users were able to use tips, while 3,035 were not. The course contained a total of 60 exercises of which 33 (55%) were in the first two weeks. For those exercises, we created 57 different tips which were referenced 242 times. Each exercise contained between 2 and 28 tips (median: 6). Across all exercises, learners sought help through a tip 67,049 times (28.31 tips per user on mean, standard deviation $\sigma = 30.72$).

Figure 1 (left) shows that learners self-identifying as beginners at course start have the highest usage ratio of tips. A decrease is observed among course participants with a higher skill level. Based on the perceived difficulty of the second week, Figure 1 (right) shows that most learners rating the exercises as very easy used fewer tips. A steady increase can be seen up to those learners rating the second week as difficult. More tips were requested during the second week (32.14% of learners, $\sigma = 11.66\%$) than in the first week (15.05% of learners, $\sigma = 6.98\%$). When learners had access to tips, their mean working time slightly increased (Welch Two Sample t-test $t = -2.6$, $p = 0.005$) by 2.7%. Learners using tips required 68% more, students not using tips about 20% less time than all users.

An analysis of all students enrolled in the course showed that a higher skill level reduced the time required to complete an

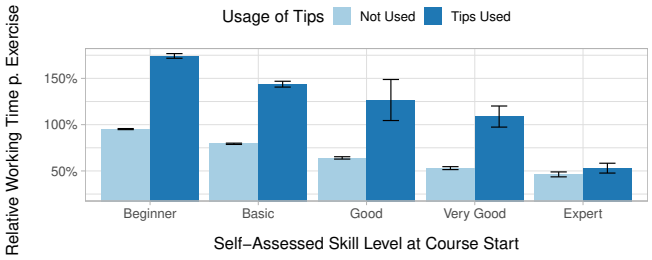

Figure 2. Comparison of the relative working time that learners required for solving a given exercise in the first half of the course depending on their skill level and the usage of tips (N = 97,851). A value of 100% represents the mean working time by students to solve the exercise.

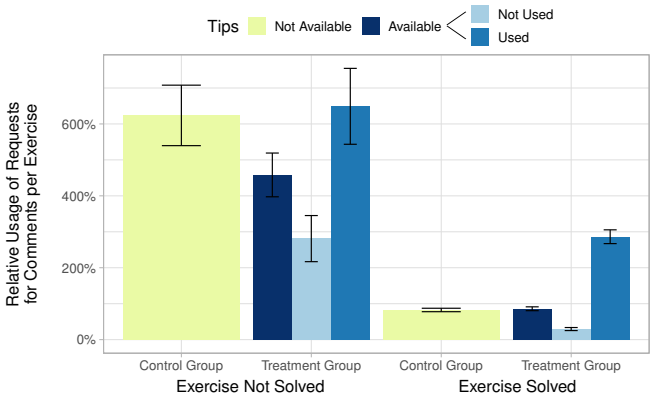

Figure 3. Comparison of the relative number of *Requests for Comments* (*RfCs*) created by learners, their A/B testing group, the usage of tips and the final status of an exercise (N = 134,102). A value of 100% represents the mean number of *RfCs* created by students per exercise. For the treatment group, the figure shows the weighted mean of *RfCs* as "Tips Available" consisting of learners either using a tip or not.

exercise. Further, Figure 2 illustrates which effects the usage of tips can have on the working time. In particular, beginners revealing tips require significantly longer to solve an exercise.

All learners had access to *Requests for Comments* (*RfCs*) allowing them to get help from peers. The mere availability of tips did not impact the usage of *RfCs* (Welch Two Sample t-test $t = -0.47$, $p = 0.32$). As visualized in Figure 3, learners solving an exercise did not show any change in requesting comments by the availability of tips (mean: 82.52% control and 85.9% treatment group). Only those learners not solving the assignment used less *RfCs* when tips were available (458.2% and 623.7% respectively). Course participants not solving an exercise are much more likely to seek help through *RfCs* (537.5%) than learners finding a solution (84.2%).

In addition to the solving state of an exercise, Figure 3 relates the usage of tips to the relative number of *RfCs*. Course participants finding a solution for an exercise request 9.7 times more comments when using tips compared to students solving an assignment without any tip (mean: 286.3% vs. 29.5%).

According to our evaluation, neither the availability nor the usage of contextual tips was able to increase the completion rate. Moreover, the data shows no significant difference in the number of code executions performed by students.

Discussion

Our results are a first evaluation of contextual tips in a programming MOOC and allow drawing some conclusions. As shown in Figure 1 (left), our primary target group of novices used tips more often than experts to seek assistance. Beginners also spent more time on a given exercise before completing the given task (Figure 2). The higher usage of tips by learners with a lower skill level might indicate that beginners require more help with the given exercises than experts.

Comparing the usage of tips in the first and second week, we observed an increase in the mean number of tips used per exercise. Learners perceived the second week as more challenging and revealed more tips. We assume that the indicated reduction of tips used by learners expressing major difficulties with the content is caused by the low participation of these users in the survey and potential course drop-outs.

For learners finishing an exercise, we conclude that the mere availability of tips had no major impact on the usage of *RfCs*. The high number of *RfCs* by learners of our treatment group receiving tips indicates that those learners are likely to use all available assistance to finally solve an exercise. Analyzing the usage of tips among learners not completing an exercise indicated a reduced usage of *RfCs* when they had access to tips. We suppose that for those learners, tips answered some of the questions that arose while working on the exercise.

An analysis of the working times per skill level and the usage of tips allow us to conclude that those users having issues with an exercise and, therefore, taking longer to solve it, were more inclined to reach out for additional assistance. Hence, reading, understanding, and applying the tip further took time leading to an overall increased working time for these students.

While we were not able to prove that the availability of tips has an impact on the completion rate of course participants, our data shows that we were able to reach our primary target group of novices with the introduction of contextual hints.

FUTURE WORK

The majority of hints used in the Python course focussed on helping students to identify the most suitable concept for a given problem and remember the correct syntax. Therefore, we focused on cheat sheet-inspired tips providing syntax information in this paper. In upcoming research, we want to compare the impact of different approaches for tips and plan to include more exercise-specific tips, similar to next-step hints. In addition, we aim to increase the perceived value and usefulness of tips with additional information.

Further, our survey results suggest that the discoverability of relevant tips within *CodeOcean* can be enhanced and might benefit from additional focus. Finally, we plan to integrate tips to just-in-time interventions or suggest learners a specific hint depending on the result of the code execution and test output.

CONCLUSION

In this paper, we evaluated the impact of contextual tips available to learners within a web-based programming environment.

We conducted an A/B testing in a regular Python course with 6,067 learners and offered tips for our treatment group. Based on our evaluation, we answer our research questions:

RQ1. How do tips influence the help-seeking behavior?

The introduction of contextual tips does not negatively affect the usage of peer-to-peer help systems. Instead, learners showed a higher chance of requesting comments when they also used tips. Some participants not finishing an exercise reduced the number of *Requests for Comments* when tips were available. This observation might suggest that tips were able to answer upcoming questions for them. Throughout the first half of the course, nearly 25% of learners regularly revealed our contextual hints while implementing an exercise.

RQ2. Which learners profit more from tips than others?

Learners self-identifying as beginners are the user group that benefits most from contextual tips. The higher the skill level at the beginning of a course is, the fewer tips are used. Our research emphasizes that more challenging exercises increase the learners' need for additional hints.

RQ3. Do tips have an impact on key metrics such as the completion rate, working times, or scores?

According to our evaluation, the mere availability of tips slightly increases the mean working time for learners by 2.7%. In particular, beginners using tips to get support spend more time within the exercise and are more likely to use other assistance features. Besides that, neither an impact of tips on the completion rate nor the scores of participants was discernible.

Overall, our findings highlight that tips are valued by novices as a relevant part of their help-seeking behavior. Contextual tips are an additional offer not impeding existing assistance features. Answers from subsequent surveys indicate the great potential tips can have and motivate us to continue researching the impact of tips. The introduction of tips as presented throughout this paper supports students to get contextual assistance within the learning environment of a MOOC.

REFERENCES

[1] Paolo Antonucci, Christian Estler, Durica Nikolić, Marco Piccioni, and Bertrand Meyer. 2015. An Incremental Hint System For Automated Programming Assignments. In *Proceedings of the 2015 ACM Conference on Innovation and Technology in Computer Science Education*. ACM, Vilnius Lithuania, 320–325. DOI:http://dx.doi.org/10.1145/2729094.2742607

[2] Kyle J. Harms, Jason Chen, and Caitlin L. Kelleher. 2016. Distractors in Parsons Problems Decrease Learning Efficiency for Young Novice Programmers. In *Proceedings of the 2016 ACM Conference on International Computing Education Research*. Melbourne, Australia, 241–250. DOI: http://dx.doi.org/10.1145/2960310.2960314

[3] Ville Isomöttönen, Antti-Jussi Lakanen, and Vesa Lappalainen. 2011. K-12 Game Programming Course Concept Using Textual Programming. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education - SIGCSE '11*. Dallas, TX, USA, 459. DOI:http://dx.doi.org/10.1145/1953163.1953296

[4] Hieke Keuning, Johan Jeuring, and Bastiaan Heeren. 2019. A Systematic Literature Review of Automated Feedback Generation for Programming Exercises. *ACM Transactions on Computing Education* 19, 1 (Jan. 2019), 1–43. DOI:http://dx.doi.org/10.1145/3231711

[5] Ada S. Kim and Andrew J. Ko. 2017. A Pedagogical Analysis of Online Coding Tutorials. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. Seattle, WA, 321–326. DOI:http://dx.doi.org/10.1145/3017680.3017728

[6] Sándor Király, Károly Nehéz, and Olivér Hornyák. 2017. Some Aspects of Grading Java Code Submissions in MOOCs. *Research in Learning Technology* 25 (2017), 16. DOI:http://dx.doi.org/10.25304/rlt.v25.1945

[7] Samiha Marwan, Nicholas Lytle, Joseph J. Williams, and Thomas W. Price. 2019. The Impact of Adding Textual Explanations to Next-Step Hints in a Novice Programming Environment. In *Proceedings of the 2019 ACM Conference on Innovation and Technology in Computer Science Education*. Aberdeen UK, 520–526. DOI:http://dx.doi.org/10.1145/3304221.3319759

[8] Silvia Muller, Monica Babes-Vroman, Mary Emenike, and Thu D. Nguyen. 2020. Exploring Novice Programmers' Homework Practices: Initial Observations of Information Seeking Behaviors. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. ACM, Portland OR USA, 333–339. DOI: http://dx.doi.org/10.1145/3328778.3366885

[9] Thomas W. Price, Yihuan Dong, Rui Zhi, Benjamin Paaßen, Nicholas Lytle, Veronica Cateté, and Tiffany Barnes. 2019. A Comparison of the Quality of Data-Driven Programming Hint Generation Algorithms. *International Journal of Artificial Intelligence in Education* 29, 3 (Aug. 2019), 368–395. DOI: http://dx.doi.org/10.1007/s40593-019-00177-z

[10] Sebastian Serth, Ralf Teusner, Jan Renz, and Matthias Uflacker. 2019. Evaluating Digital Worksheets with Interactive Programming Exercises for K-12 Education. In *IEEE Frontiers in Education (FIE)*. Cincinnati, USA. DOI:http://dx.doi.org/10.1109/FIE43999.2019.9028680

[11] Thomas Staubitz, Hauke Klement, Ralf Teusner, Jan Renz, and Christoph Meinel. 2016. CodeOcean - A Versatile Platform for Practical Programming Excercises in Online Environments. In *IEEE Global Engineering Education Conf. (EDUCON)*. Abu Dhabi, 314–323. DOI:http://dx.doi.org/10.1109/EDUCON.2016.7474573

[12] Ralf Teusner, Thomas Hille, and Thomas Staubitz. 2018. Effects of Automated Interventions in Programming Assignments: Evidence from a Field Experiment. In *Proceedings of the Fifth Annual ACM Conference on Learning at Scale - L@S '18*. London, UK. DOI: http://dx.doi.org/10.1145/3231644.3231650