Prompting for Free Self-Explanations Promotes Better Code Comprehension

Vasile Rus
Department of Computer Science
Institute of Intelligent System
University of Memphis
Memphis, TN, USA
emailvrus@memphis.edu

Kamil Akhuseyinoglu
School of Computing and Information
University of Pittsburgh
Pittsburgh, PA, USA
emailKAA108@pitt.edu

Jeevan Chapagain
Department of Computer Science
Institute of Intelligent System
University of Memphis
Memphis, TN, USA
emailjchpgain@memphis.edu

Lasang Tamang
Department of Computer Science
Institute of Intelligent System
University of Memphis
Memphis, TN, USA
emailljtamang@memphis.edu

Peter Brusilovsky
School of Computing and Information
University of Pittsburgh
Pittsburgh, PA, USA
emailpeterb@pitt.edu

ABSTRACT

We present in this paper a summary analysis of log files collected during an experiment designed to test the hypothesis that prompting for free self-explanations leads to better comprehension of computer code examples. Indeed, the results indicate that students who were prompted to self-explain while trying to understand code examples performed significantly better at predicting the correct output of the examples than students who were just prompted to read the code examples and predict their output.

Keywords

self-explanation, code comprehension

1. INTRODUCTION

Code comprehension, i.e., understanding of computer code, is a critical skill for both learners and professionals. Students learning computer programming spend a significant portion of their time reading or reviewing someone else's code (e.g., code examples from a textbook or provided by the instructor). Furthermore, it has been estimated that software professionals spend at least half of their time analyzing software artifacts in an attempt to comprehend computer source code. Reading code is the most time-consuming activity during software maintenance, consuming 70% of the total life-cycle cost of a software product [20]. O'Brien [20] notes that source code comprehension is required when a programmer maintains, reuses, migrates, re-engineers, or enhances software systems. Therefore, offering support to enhance learners' source code comprehension skills will have lasting positive effects for their academic success and future professional careers.

In fact, such support should have a significant impact on aspiring Computer Science (CS) students' learning, self-efficacy, and overall success and graduation rates. Indeed, although computing skills are in high-demand, and the number of aspiring CS students is encouraging, a large gap between the supply of CS graduates and the demand persists because college CS programs have attrition rates of 30–40% (or even higher) in introductory CS courses (e.g., CS1 and CS2) [13, 2]. Advances towards the development of effective and engaging instructional interventions to improve comprehension and learning in introductory Computer Science courses at the college level, to reduce attrition rates and increase retention, and to ultimately produce more and better trained graduates are much needed. The result will be a win-win-win situation for aspiring students, CS programs and their organizations, and the overall economy.

Our broader research agenda is to propose and study such novel instructional interventions to improve comprehension and learning in intro-to-programming courses at college level. Specifically, our goal is to develop a web-based learning and

research environment that allows us to model, monitor, scaffold, and investigate source code comprehension and learning processes. A sub-goal is to explore instructional strategies that promote deep code comprehension and learning. One such strategy, based on theories of self-explanation, is to elicit self-explanations, i.e., student self-generated explanations of a target text, e.g. science paragraph from a science textbook, or, in our case, of a code example. To this end, we present in this paper a summary of a randomized controlled experiment in which we tested the hypothesis that prompting for free-self explanation, i.e., self-explanations that students freely generate without any significant training.

2. RELATED WORK

Prior CS education research documented the difficulty novice programmers face with constructing accurate mental models during key learning activities, such as source code comprehension [26, 21, 22, 16]. This challenge is not surprising given that constructing mental representations is considered a higher-level skill of comprehension, typically engendering a high cognitive load [14, 27, 25, 12]. The importance of building accurate mental models during learning tasks has been well established for decades in domains like science [7, 11, 19, 10] as well as in CS education [26, 21, 22, 16]. Nevertheless, further research is needed to fully understand what factors can mediate the construction of accurate mental model and learning and build effective instructional interventions that can monitor and scaffold learners' comprehension and learning processes. The intervention can be instructor driven or Artificial Intelligence driven like in adaptive instructional systems of the kind we intend to develop.

Self-explanation theories [5, 4] indicate that students who engage in self-explanations, i.e. explaining the target material to themselves, while learning are better learners, i.e. learn more deeply and show highest learning gains. The positive effect of self-explanation on learning has been demonstrated in different science domains such as biology [6] and physics [9], math [1], and programming [3]. Self-explanation's effectiveness for learning is attributed to its constructive nature, e.g., it activates several cognitive processes such generating inferences to fill in missing information and integrating new information with prior knowledge, monitoring and repairing faulty knowledge, and its meaningfulness for the learner, i.e., self-explanations are self-directed and self-generated making the learning and target knowledge more personally meaningful, in contrast to explaining the target content to others [23]. Several types of self-explanation prompts have been identified and explored such as justification-based self-explanation prompts [8] and meta-cognitive self-explanation prompts [6].

Indeed, there are different ways to elicit self-explanations which result in different types of self-explanations such as spontaneous self-explanations (no prompting), free or open-ended self-explanations (simple prompting to self-explain), guided (see the Socratic method description later), and scaffolded self-explanations (in this case students are encouraged to self-explain as much as possible by themselves and offered support in the form of hints when floundering). Other forms of self-explanations have been tried such as "complete given self-explanations (fill-in the blank self-explanations)" [15] and "select a self-explanation/menu-based self-explanations" [1] which one may argue are not true self-explanations as the learner does not generate the explanation, i.e., the 'self' part of the 'self-explanation' is missing. Furthermore, self-explanation prompts can emphasize various aspects of self-explanations resulting in justification-based self-explanation prompts [9] or meta-cognitive self-explanation prompts [6]. Self-explanations can also be categorized based on being spoken (or thinking out loud) versus typed or written. The former can be regarded as reflecting more directly students thinking process whereas the latter may represent a more refined version of their thinking process as when writing we have a tendency to refine our sentences and therefore one can argue the typed self-explanation engage some reflection and refinement processes. Both have been studied in the literature for other domains, e.g., [17] studied think aloud self-explanations of instructional materials in the context of science learning and science text comprehension, whereas [18] explored the role of written self-explanations for reading comprehension of scientific texts and learning of target concepts or trying to solve a problem. They found that this type of self-explanation benefits proficient readers in general compared to less proficient readers [18].

We explore here the role of prompting for typed, free self-explanations during comprehension and learning tasks. While students received no significant training with respect to what a self-explanation should look like, e.g., integrating new information with prior knowledge through bridging inferences, they were shown an ideal self-explanation for one code example at the beginning of the experiment because our pilot experiments demonstrated that most students just "translate in plain English" when prompted for self-explanations of code, i.e., they restate in words each line of code with no inferencing about the higher level functionality of various blocks of code. To measure the accuracy of the constructed mental models for given code example, we ask students' to predict the output of each of the code examples. As control, we use a condition in which students are asked to read and predict the output of the code examples (no self-explanation were asked for).

3. EXPERIMENTAL SETUP

We conducted a randomized control trial experiment in which participants were assigned to two approximately equal experimental groups: a free Self-Explanation group and a Prediction Only (control) group. To balance the number of participants in each group, we used a group balancing approach which kept half students in one group while other half students in second group. Participants were debriefed about the purpose of the experiment and given an informed consent form which participants had to read and sign if they agreed with it. Those who agreed to participate in the experiment were given clear instructions by the experimenter and a quick introduction to the interface of the experimental system which was accessed through a browser.

The experiment consisted of students answering a brief background questionnaire regarding their programming experience, a more specific programming knowledge self-efficacy questionnaire, a confidence survey targeting the programming concepts/topics covered by the main task, a pre-test assessing their prior knowledge of the topics covered in the main task, a self-efficacy survey, the main task which in-

volved understanding 6 Java code examples and predicting their output, a 1-minute break to allow students to recover after the main task, a post-test assessing their knowledge on the same topics, and post main task self-efficacy survey. The programming concepts that we tried to cover during our experiment are: operator precedence, nested if-else, for loops,while loops,arrays,creating objects and using their methods.

In the main task, students were shown 6 Java code examples and were asked to read in order to understand what they do and then based on their understanding predict the output of the code examples. When predicting the output, they were also asked to indicate their confidence. The correct output to each Java code example was shown immediately after they entered their prediction. In the Self-Explanation condition, they were asked to self-explain while reading the code. The following self-explanation instructions were given: in their own way what does that code block do to collect the self-explanations from them.

> When you read a Java code example in order to comprehend it, like the JAVA code below, you are supposed to read each line of the code carefully and while doing so explain what the code means to you. While self explaining, you may want to think of:
>
> What new information does each line provide for you? How does it relate to what you've already read? Does it give you a new insight into your understanding of how the code works? or does it raise a question in your mind?
>
> Describe whatever is going through your mind - even if it seems unimportant. You may need to go back and re-read parts of the code to really understand the whole code. Some people find it helpful, when reading difficult material, to draw a picture or take notes on a piece of paper. Please feel free to do what is best for you. Once you finished reading the whole code, please explain briefly what the overall goal of the code is, e.g., generating a Bingo board, and predict its output. Keep in mind that there are no "right" or "wrong" self-explanations.

We have been looking for self-explanation instructions in the literature and with few exceptions they were not mentioned. The instructions can have a significant impact on the outcome of the experiment which is the reason we show them here as seen by the participants. We believe any future attempts to reproduce our results should use the above instructions verbatim.

Then, participants were shown 6 code examples and asked to either self-explain what the code does (free Self-Explanation) or just predict the output of the code examples (Prediction Only). As noted before, all participants took a pretest before being being assigned to an experimental condition and posttest afterwards. The pre-/post-test scores were used to calculate learning gains as a measure for the effectiveness of the interventions. We used a web-based software system to run the experiment.

We conducted a t-test to compare the mean learning gains of the experimental groups. Furthermore, we categorized participants into low and high-prior knowledge groups based on their mean pretest score for both the free Self-Explanation and the Socratic method groups separately. Then, we conducted an independent sample t-test to compare learning gains between the low and high prior knowledge groups. More details about group design, participants, materials used, experiment protocol and measures are given in the following sections. It should be noted that all interactions between each participant and the system were logged in an anonymous manner and have been used to do the post experiment analyses presented later.

3.1 Group Design

Participants were randomly assigned to two different groups i.e. first group were shown a model self-explanation of a code example and then prompted to self-explain 6 Java code examples and then predict the output of each of the code examples while the second group had to predict the output by only reading in order to understand each of the code examples. Out of the 39 students involved, 20 participants were assigned to the first group and 19 participants in the second group.

3.2 Participants

While our intention initially was to recruit participants from the intro-to-programming classes (CS0, CS1, and CS2), due to low recruitment rates from those courses we expanded our pool of subjects to all undergraduates and graduate students. The low recruitment rates from the CS0, CS1, and CS2 courses was low due primarily to the timing of our experiment which was at the end of the Spring semester, right before the final exams. Other factors contributed as well such as the whole COVID-19 (corona virus) situation because of which all experimental session were run fully online. We ended up recruiting and fully running 39 participants from two US universities of which 36 were undergraduate students and 3 graduates students. The undergraduate participants were attending the following courses: CS1 (4 students), CS2(21), Data Structures - also called CS3 by some (8), and CS 3351???? (1). All participants were familiar with the JAVA programming language.

3.3 Materials

Participants in each of the two experimental conditions were shown same set of 6 source code examples, i.e., we controlled for content but not for time which as we will see later in the experimental results analysis had an impact in terms of tiring effects for the Self-Explanation group of participants. We call the 6 code examples the main task. As mentioned before, participants took a pre-test as well that consisted of 6 code examples matching in terms of content, i.e., target concepts, the code examples in the main task. Furthermore, participants took a post-test consisting of 6 code examples matching in content that examples in the main task and pre-test. For the pre-test and post-test, learners were supposed to just provide the predicted output. As already noted, the pre-test and post-test were not identical but they were equivalent in term of concepts tested and difficulty level. The main programming concepts covered by the experiment were: operator precedence, nested $if-else$,

for loops, *while* loops, arrays, creating objects and using their methods. Each of these concepts were present in the in the code examples used in the pre-test, post-test, and the main task.

4. RESULTS

As noted earlier, the experimental system logged all student responses to various system prompts including time stamps associated with various student actions. The log files have been the basis of the analyses presented in this section. The analyses presented here are just a subset of the type of data analyses or data mining that can be done on the logged experimental data.

As a reminder, the key research question or goal of the experiment was to see if prompting for free self-explanation helps source code comprehension. This can be measured by the participant's performance on the main comprehension tasks: the average score was 5.05 for Self-Explanation group while the Prediction-only group had an average score of 4.1 for correctly predicting the output of each of the 6 Java code examples they were supposed to read and comprehend. The accuracy of the predicted output on the main comprehension tasks is a direct reflection of the accuracy of the mental models students constructed during the reading of those code examples. The results in Table 1 shows us that there is an statistical significant difference between Self-Explanation group (M = 5.05, SD = 1.129) and Prediction group (M =4.10 and SD = 1.410) in the main task. The magnitude of the difference in the means (mean difference = -0.953, 95% confidence interval: -1.784 to -0.121) is large (Cohen's d = 0.199) as suggested by [24]. Based on these results, we conclude that prompting for typed, free self-explanations leads to better code comprehension.

As a next step and as a way to better understand the main result of the experiment outlined above, we explored whether prior knowledge, as measured by the pre-test, can be a mediating factor and if it differs among the experimental groups.

Table 1: Independent sample t-test for main task of self explanation and prediction group

Group	N	Mean	SD	t-val	Sig.
Self Explanation	19	5.05	1.129	-2.326	0.026
Prediction	20	4.10	1.410		

Overall, participants had an average pre-test score of 4.84 for the Self-Explanation group and 4.75 for the Prediction group. A t-test, which met all the standard assumptions (continuous scale for dependent variable, random sampling, independence of observations, normal distribution and homogeneity of variance), indicated that the two groups i.e. Self-Explanation and Prediction, are equivalent in term of prior knowledge. The result of the t-test can be seen in Table 2.

Once we showed the groups are equivalent, we performed an independent t-test between the predictions score of self-explanation and prediction group to compare their scores.

We also performed analysis of covariance (ANCOVA) by taking experimental condition as grouping factor and pre-test score as covariate which also resulted in a significant difference. The Self-Explanation group performed better than the Prediction-only group (5.05 - 4.10)/6 * 100 = 15.83 %.

Table 2: Independent sample t-test for pretest of self explanation and prediction group

Group	N	Mean	SD	t-val	Sig.
Self-Explanation	19	4.84	1.537	-0.212	0.834
Prediction	20	4.75	1.164		

Given the positive impact of prompting for typed, free self-explanations, we wondered if prior knowledge is a mediating factor for participants in the Self-Explanation group. That is, we wondered if higher prior knowledge leads to better self-explanations and better comprehension. To this end, we divided the students in the Self-Explanation group in two subgroups: high prior knowledge versus low prior knowledge. We used the mean pre-test score of 4.84 as the splitting point. We are aware that using the mean score to obtain the two subgroups may lead to some students in the the two subgroups having very similar scores, e.g., those participants with scores close to the mean score. However, using a top and bottom quartile split, which would eliminate this issue of students in the different subgroups having similar scores, would have led to small subgroups given that our overall $n = 39$ participants.

Based on the main comprehension task performance, there was a significant difference between the high prior knowledge and low prior knowledge subgroups. Table 3 shows us that there is an statistical significant difference between main tasks of self-explanation group with low prior knowledge (M = 6.00, SD = 0.00) and high prior knowledge (M = 4.692 and SD = 1.182). The magnitude of the difference in the means (mean difference = 1.307, 95% confidence interval: 0.165 to 2.449) is large (Cohen's d = 0.498).

Table 3: Independent sample t-test for main task of self explanation group based on low prior knowledge and high prior knowledge

Group	N	Mean	SD	t-val	Sig.
Low Prior	5	6.00	0.00	3.989	0.002
High Prior	13	4.692	1.182		

5. CONCLUSIONS

We presented in this paper the results of a series of analyses of log data from a randomized controlled trial experiment designed to test the hypothesis that an instructional strategy that prompts for typed, free self-explanations can help code comprehension. The experimental results obtained do indeed support the hypothesis. Furthermore, we found out that students with lower prior knowledge are much more helped by prompting for typed, free self-explanations. That is, this strategy seems to be particularly useful for low prior knowledge students.

We do plan to further analyze the results of the experiment we conducted. For instance, we would like to analyze the mental models students constructed by annotating the self-explanations along a number of 10 dimensions inspired by

theories of self-explanation and code comprehension as proposed by Lasang and et al. (2021). Furthermore, we intend to conduct more experiments to understand what other factors mediate the quality of the mental models constructed such as the readability of the code being red. In the current experiment, the code examples followed professional code writing style and therefore could be deemed as highly readable. However, there were no comments in those code examples which could further increase the readability of the examples. However, the presence of comments may give some students the illusion of understanding by demotivating them to read carefully the code and be tempted to just read the comments thus leading to less effective comprehension processes. There is much work to be done and we are excited to further the very promising work presented here.

Acknowledgements

This work was supported by the National Science Foundation under award 1822816. All findings and opinions expressed or implied are solely the authors'.

6. REFERENCES

[1] V. A. Aleven and K. R. Koedinger. An effective metacognitive strategy: Learning by doing and explaining with a computer-based cognitive tutor. *Cognitive science*, 26(2):147–179, 2002.

[2] T. Beaubouef and J. Mason. Why the high attrition rate for computer science students: some thoughts and observations. *ACM SIGCSE Bulletin*, 37(2):103–106, 2005.

[3] K. Bielaczyc, P. L. Pirolli, and A. L. Brown. Training in self-explanation and self-regulation strategies: Investigating the effects of knowledge acquisition activities on problem solving. *Cognition and instruction*, 13(2):221–252, 1995.

[4] M. T. Chi. Self-explaining expository texts: The dual processes of generating inferences and repairing mental models. *Advances in instructional psychology*, 5:161–238, 2000.

[5] M. T. Chi, M. Bassok, M. W. Lewis, P. Reimann, and R. Glaser. Self-explanations: How students study and use examples in learning to solve problems. *Cognitive science*, 13(2):145–182, 1989.

[6] M. T. Chi, N. De Leeuw, M.-H. Chiu, and C. LaVancher. Eliciting self-explanations improves understanding. *Cognitive science*, 18(3):439–477, 1994.

[7] M. T. Chi, P. J. Feltovich, and R. Glaser. Categorization and representation of physics problems by experts and novices. *Cognitive science*, 5(2):121–152, 1981.

[8] C. Conati, J. Larkin, and K. VanLehn. A computer framework to support self-explanation. In *Proceedings of AI-ED 97 World Conference on Artificial Intelligence in Education*, volume 39, pages 279–276. Citeseer, 1997.

[9] C. Conati and K. VanLehn. Further results from the evaluation of an intelligent computer tutor to coach self-explanation. In *Int. Conference on Intelligent Tutoring Systems*, pages 304–313. Springer, 2000.

[10] T. de Jong and M. G. Ferguson-Hessler. Knowledge of problem situations in physics: A comparison of good and poor novice problem solvers. *Learning and Instruction*, 1(4):289–302, 1991.

[11] A. A. DiSessa. Toward an epistemology of physics. *Cognition and instruction*, 10(2-3):105–225, 1993.

[12] A. C. Graesser and D. S. McNamara. Computational analyses of multilevel discourse comprehension. *Topics in cognitive science*, 3(2):371–398, 2011.

[13] M. Guzdial and E. Soloway. Teaching the nintendo generation to program. *Communications of the ACM*, 45(4):17–21, 2002.

[14] W. Kintsch. The role of knowledge in discourse comprehension: A construction-integration model. *Psychological review*, 95(2):163, 1988.

[15] K. Kwon, C. D. Kumalasari, and J. L. Howland. Self-explanation prompts on problem-solving performance in an interactive learning environment. *Journal of Interactive Online Learning*, 10(2), 2011.

[16] L. E. Margulieux, M. Guzdial, and R. Catrambone. Subgoal-labeled instructional material improves performance and transfer in learning to develop mobile applications. In *Proceedings of the ninth annual international conference on International computing education research*, pages 71–78, 2012.

[17] D. S. McNamara and J. P. Magliano. Self-explanation and metacognition: The dynamics of reading. In *Handbook of metacognition in education*, pages 72–94. Routledge, 2009.

[18] B. Muñoz, J. P. Magliano, R. Sheridan, and D. S. McNamara. Typing versus thinking aloud when reading: Implications for computer-based assessment and training tools. *Behavior research methods*, 38(2):211–217, 2006.

[19] M. J. Nathan, W. Kintsch, and E. Young. A theory of algebra-word-problem comprehension and its implications for the design of learning environments. *Cognition and instruction*, 9(4):329–389, 1992.

[20] M. P. O'Brien. Inference-based comprehension and expectation based processing in program comprehension. In *9th International Workshop on Program Comprehension*, pages 71–78, 2001.

[21] N. Pennington. Comprehension strategies in programming. In *Empirical Studies of Programmers: Second Workshop, 1987*, pages 100–113, 1987.

[22] V. Ramalingam, D. LaBelle, and S. Wiedenbeck. Self-efficacy and mental models in learning to program. In *Proceedings of the 9th annual SIGCSE conference on Innovation and technology in computer science education*, pages 171–175, 2004.

[23] M. Roy and M. T. Chi. The self-explanation principle in multimedia learning. *The Cambridge handbook of multimedia learning*, pages 271–286, 2005.

[24] S. S. Sawilowsky. New effect size rules of thumb. *Journal of Modern Applied Statistical Methods*, 8(2):26, 2009.

[25] C. Snow. *Reading for understanding: Toward an R&D program in reading comprehension*. Rand Corp., 2002.

[26] E. Soloway and K. Ehrlich. Empirical studies of programming knowledge. *IEEE Transactions on software engineering*, (5):595–609, 1984.

[27] R. A. Zwaan and G. A. Radvansky. Situation models in language comprehension and memory. *Psychological bulletin*, 123(2):162, 1998.