A Vision for Interactive Suggested Examples for Novice Programmers

Michelle Ichinco
Department of Computer Science
University of Massachusetts Lowell
Lowell, MA, USA
michelle_ichinco@uml.edu

***Abstract*—Many systems aim to support programmers within a programming context, whether they recommend API methods, example code, or hints to help novices solve a task. The recommendations may change based on the user's code context, history, or the source of the recommendation content. They are designed to primarily support users in improving their code or working toward a task solution. The recommendations themselves rarely provide support for a user to interact with them directly, especially in ways that benefit the knowledge or understanding of the user. This poster presents a vision and preliminary designs for three ways a user might learn from interactions with suggested examples: describing examples, providing detailed relevance feedback, and selective visualization and tinkering.**

***Index Terms*—Novice programmers, interactive suggestions, example code**

I. Introduction

Many systems suggest example code or support novices in finding example code relevant to their programs, both in task-based contexts as well as when programmers design their own projects [1]–[3]. Beyond a description or highlighting of the relevant elements, the examples typically provide little support for learning [4], [5]. Many novice programmers begin to learn independently, in non-task contexts, like by creating their own app, website, or game. Novices in these contexts thus need more support from suggestion-based systems to actually learn from examples. Research in cognitive load theory provides approaches for increasing learning gains from educational material. This poster presents designs for three interaction techniques for suggested examples based on cognitive load theory.

II. Cognitive Load Theory

Cognitive load theory is a theory often used in the design of instructional material in order to support learning [6]. It is often associated with 'worked examples', which are examples with worked out steps, usually for topics like mathematics or physics. Humans have limited cognitive load to spend at any point in time. Cognitive load theory provides methods of reducing extraneous cognitive load, which interferes with learning, and increasing germane cognitive load, which supports deep learning. This vision incorporates three elements of cognitive load theory into the design of interaction methods: self-explanation, multiple examples, and fading.

978-1-5386-4235-1/18/$31.00 ©2018 IEEE

Self-explanation causes learners to produce explanations of new material and relate those explanations to the general principles being taught [7]. This process deepens learners' understanding of the new material. Recent work has shown that self-explanation can encourage learners to create labels for programming examples [8], [9]. Ideally, encouraging novices to author descriptions of code examples will result in the benefits of effective self-explanation.

Self-explanation can be especially helpful for the comparison of multiple examples. Research has found that providing multiple worked examples can help learners [10]. However, Catrambone and Holyoak found that multiple examples only support learners in problem solving when the learners explore the similarities between the examples [11]. Having learners explain the relevance or lack of relevance between their code and examples, will likely have similar effects to the presentation of multiple worked examples combined with self-explanation.

While self-explanation increases germane cognitive load, faded worked examples reduce the amount of new information a learner needs to deal with at one time. Fading involves a sequence of worked examples. In each subsequent worked example, support is removed [12]. Thus, fading supports learners by reducing the extraneous cognitive load. Researchers have shown that faded worked examples can be effective for programming [13]. Our third interaction method, selective visualization and tinkering, aims to approximate fading by enabling novices to focus on smaller pieces of an example, rather than attempting to understand how the entire example works at one time.

III. Suggestion Interaction Types

This poster presents three potential interaction methods: 1) describing code examples, 2) providing detailed relevance feedback, and 3) selective visualization and tinkering. Each section describes why this interaction method should support learning and why users will likely be motivated to participate in these interactive activities.

IV. Describing Code Examples and Selecting Appropriate Descriptions

Having learners describe code examples and select appropriate descriptions should elicit self-explanation of the provided examples. In order to describe an example or select a correct description, the learner will have to figure out how it works.

Other tools for learning have had learners successfully label videos [14], math problems [15], and programming worked examples [8]. These studies support the value of this method of eliciting self-explanation, but evaluate the method in controlled studies where that is a primary task. I hypothesize that this type of approach would also work in the midst of a programming task where a user can choose whether or not to participate in the example labeling. One way to motivate users to author these descriptions may be by telling them that their description will help other users. Many programmers choose to help each other, such as in the Stack Overflow forum [16]. If users are motivated by helping each other, they will likely try to author or select the best description they can.

This process of describing code examples will likely encourage learners to think more deeply about examples. We also want to encourage learners to think deeply about why they received feedback and how it relates to their own code by encouraging them to provide detailed relevance feedback.

V. Providing Detailed Relevance Feedback

Thinking critically about the relevance of a suggested example to a user's code will likely deepen their understanding of the example and their code. Compared to a quick up or down vote like in many existing systems, providing detailed feedback would hopefully encourage a learner to spend time thinking about how their code is related or not related to the suggested example. As a side benefit, these descriptions can also help the system designers to improve the relevance of suggestions or evaluate a learner's understanding. Leveraging this fact may encourage learners to provide detailed descriptions, as the better their descriptions are, the better the support system can help them.

VI. Selective Visualization and Tinkering

Current example code suggestions do not typically enable users to tinker with or explore examples without inserting them into the user's code. Some allow execution of the entire code snippet, along with a visualization of the entire code snippet output, but do not allow changes or selecting a specific part of the code to execute [4]. When watching the execution of a whole code snippet, it might be hard, especially for a novice, to determine which elements of code have which effect. Enabling a user to tinker with an example without inputting it into their code might enable them to better understand each element of a code example before they try to implement it themselves. This could prevent novices from creating new errors when using new code.

VII. Future Work

This poster presents a vision and preliminary designs for interactive suggested examples. The ideas presented require iteration and evaluation with users. User studies will reveal whether users' motivations match the provided interaction opportunities and whether these interaction methods improve learning.

References

[1] J. Cao, Y. Riche, S. Wiedenbeck, M. Burnett, and V. Grigoreanu, "End-user mashup programming: through the design lens," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2010, pp. 1009–1018. [Online]. Available: http://dl.acm.org/citation.cfm?id=1753477

[2] T. W. Price, Y. Dong, and D. Lipovac, "iSnap: towards intelligent tutoring in novice programming environments," in *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education*. ACM, 2017, pp. 483–488.

[3] M. Ichinco and C. Kelleher, "Towards block code examples that help young novices notice critical elements," in *2017 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, Oct. 2017, pp. 335–336.

[4] M. Ichinco, W. Y. Hnin, and C. L. Kelleher, "Suggesting API Usage to Novice Programmers with the Example Guru," in *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, ser. CHI '17. New York, NY, USA: ACM, 2017, pp. 1105–1117. [Online]. Available: http://doi.acm.org/10.1145/3025453.3025827

[5] B. Hartmann, D. MacDougall, J. Brandt, and S. R. Klemmer, "What would other programmers do: suggesting solutions to error messages," in *Proc. 28th int. conf. on Human factors in computing systems*, 2010, pp. 1019–1028.

[6] J. Sweller, "Cognitive load theory, learning difficulty, and instructional design," *Learning and instruction*, vol. 4, no. 4, pp. 295–312, 1994.

[7] 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*, vol. 13, no. 2, pp. 145–182, 1989.

[8] B. B. Morrison, L. E. Margulieux, and M. Guzdial, "Subgoals, context, and worked examples in learning computing problem solving," in *Proceedings of the eleventh annual international conference on international computing education research*. ACM, 2015, pp. 21–29.

[9] B. B. Morrison, L. E. Margulieux, B. Ericson, and M. Guzdial, "Subgoals help students solve Parsons problems," in *Proceedings of the 47th ACM Technical Symposium on Computing Science Education*. ACM, 2016, pp. 42–47.

[10] R. K. Atkinson, S. J. Derry, A. Renkl, and D. Wortham, "Learning from examples: Instructional principles from the worked examples research," *Review of educational research*, vol. 70, no. 2, pp. 181–214, 2000. [Online]. Available: http://rer.sagepub.com/content/70/2/181.short

[11] R. Catrambone and K. J. Holyoak, "Overcoming contextual limitations on problem-solving transfer." *Journal of Experimental Psychology: Learning, Memory, and Cognition*, vol. 15, no. 6, p. 1147, 1989.

[12] A. Renkl, R. K. Atkinson, and C. S. Gro\s se, "How fading worked solution steps worksa cognitive load perspective," *Instructional Science*, vol. 32, no. 1-2, pp. 59–82, 2004.

[13] S. Gray, C. St Clair, R. James, and J. Mead, "Suggestions for graduated exposure to programming concepts using fading worked examples," in *Proceedings of the third international workshop on Computing education research*. ACM, 2007, pp. 99–110.

[14] S. Weir, J. Kim, K. Z. Gajos, and R. C. Miller, "Learnersourcing subgoal labels for how-to videos," in *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*. ACM, 2015, pp. 405–416.

[15] J. J. Williams, J. Kim, A. Rafferty, S. Maldonado, K. Z. Gajos, W. S. Lasecki, and N. Heffernan, "Axis: Generating explanations at scale with learnersourcing and machine learning," in *Proceedings of the Third (2016) ACM Conference on Learning@ Scale*. ACM, 2016, pp. 379–388.

[16] "Stack overflow." [Online]. Available: http://stackoverflow.com