# Towards a Programming Environment that Adaptively Suggests Examples and Corresponding Puzzles Based on Programmer Skill

Kyle J. Harms
Department of Computer Science & Engineering
Washington University in St. Louis
St. Louis, Missouri, United States
harmsk@seas.wustl.edu

## I. Introduction

Programmers often re-appropriate code or new programming skills they find in unfamiliar code within their own programs [1], [2]. This process enables programmers, including novices, to acquire new programming knowledge while working on their own programming projects. Unfortunately, novice programmers often have difficultly understanding and integrating existing code into their own programs [3], thereby limiting acquisition of new programming concepts found within unfamiliar code.

In this paper, I describe my prior work attempting to decrease the burden of learning new programming concepts found in unfamiliar code with automatically generated programming tutorials. Later, I introduce my proposal to create a programming environment that adapts to the skill level of the programmer while introducing programming concepts found within existing code by suggesting example code alongside programming puzzles.

## II. Learning Programming with Automatically Generated Tutorials

Initially, I developed automatically generated programming tutorials that converted a code snippet into an onscreen tutorial to enable novices to learn new programming concepts found within unfamiliar code [4]. These tutorials walked a user through all of the steps necessary to reconstruct a code snippet within their own program. In an evaluation, users who reconstructed code using the tutorials performed 64% better on near transfer tasks than participants whom did not follow a tutorial [4].

While promising, these same participants only succeeded in 52% of the near transfer tasks [4]. I originally hypothesized that this was due to over-whelming participants working memory resources with many extraneous steps within the tutorials [5]. To evaluate this, I conducted a preliminary study that reduced the number of steps within the generated tutorials by removing all steps that were not related to unfamiliar programming concepts. The removal of the extra steps made no difference on performance of a near transfer task.

This suggests that simply removing extra information is not sufficient to encourage further learning of unfamiliar programming concepts. Instead, as suggested by cognitive load theory, I need to tailor new information to the specific learning needs of each programmer to improve learning.

## III. Cognitive Load Theory

Humans have very limited working memory. When learning new concepts, we process the new information in our working memory. If our working memory becomes overwhelmed, then our ability to learn additional information is severely hampered [6]. Cognitive load theory suggests that by carefully managing a learner's working memory resources we can increase the efficiency for a learner to acquire new information [6].

There are two main sources of cognitive load: intrinsic and extraneous. Intrinsic cognitive load is imposed by the intrinsic nature of the information. Whereas extraneous cognitive load is imposed by the manner in which the information is presented. Applying these principles is commonly accomplished by providing instructional materials that take into consideration a learner's prior knowledge and that are carefully authored to emphasize the new information. This helps prevent the learner's working memory resources from becoming exhausted while processing the new material.

## IV. Adapting the Programming Environment based on Programmer Skill

I plan to employ cognitive load theory to develop a programming environment that continuously adapts to the individual learning needs of each programmer. To accomplish this, I plan to suggest code examples that a programmer will find relevant to their current project but that also align with their current skill level. If a programmer then chooses to add the suggested example code into their program I will generate a code completion puzzle from the example code. The user then must successfully complete the puzzle to add the new code into their program. To adapt to the programmer's skill, I plan to additionally develop a rapid programming assessment technique that is suitable to be used inline within a programming environment.

I hypothesize that a programming environment that adaptively suggests code examples and presents corresponding completion puzzles based on a learner's assessed programming skill will enable the user to learn new programming concepts found within the unfamiliar code.

## A. Rapid Programming Assessment

Assessment of a learner's skill is typically done with a traditional test, however this often takes substantially longer than the *rapid online test method* [7]. The rapid online test is well suited to live environments because it provides an accurate assessment of a learner's knowledge and users can complete the assessment in a fraction of the time of traditional testing methods [7]. The rapid test method prompts learners to complete the next step towards the solution to the problem. Based on the granularity of the intermediate step we can infer the learner's degree of understanding for this concept [7].

The rapid online testing method is specifically designed for well structured domains like mathematics that have known and well-defined intermediate steps. In domains like programming, intermediate steps are not well-defined. I plan to create a rapid programming assessment based on the work done for another ill-structured domain, reading comprehension [8]. Here, the rapid test asks users to study an example for a short timed period, followed by one multiple choice question. Each of the answers in the question describe the previously studied example with varying degrees of abstractness. From the user's choice we can infer the user's skill level by the degree of abstractness of the chosen answer [8].

## B. Suggesting Examples

To effectively manage a programmer's cognitive load the programming environment should suggest code examples that closely align with user's current knowledge. Using the information gained from the rapid assessment the programming environment can suggest examples that will likely not overwhelm the programmer's working memory resources. This means the environment will not suggest examples that contain concepts that programmer already understands, because this may hinder learning [6]. Instead the environment will carefully select examples that go slightly beyond what the programmer currently understands and ignore examples that are likely to overwhelm the programmer's working memory resources because of their inherit difficulty.

Programming concepts are typically taught in a structured progression in traditional learning environments. Teachers will often teach programming concepts in an order that gradually builds to more advanced concepts. I also plan to develop a programming concept progression that is structured based on a similar order used in traditional learning environments. Suggesting examples based on this static progression along with the user's assessed skill level will help ensure that users are not exposed to programming concepts before they are able to fully process the material in their working memory.

There are other programming environments that also suggest examples to users [9], [10]. However, I am not aware of any systems that suggest programming example based on a learner's existing knowledge.

## C. Programming Puzzles

Once a user has decided to add a suggested example into their program, we can use this as an opportunity to emphasize new programming concepts found within the unfamiliar code. My prior work emphasized these concepts by asking users to follow a generated tutorial. These tutorials required very little active engagement, which is required for learning [6]. For learning to occur, users must process information within their working memory [6]. Completion problems and worked examples are one effective way to foster learning that are commonly used when employing cognitive load theory [6].

The programming environment will generate a completion puzzle from the suggested example that is also accompanied by several worked examples closely related to the puzzle. Worked examples enable learners to study the structure of the concept to then be able to solve their current problem. In this case a programmer will see a puzzle that contains the scrambled code from the suggested example. They can then use the corresponding worked examples as a reference to help them unscramble the code puzzle.

The puzzles themselves impose extraneous cognitive load which means I must also vary the complexity of these puzzles based on the user's prior experience. The first puzzle a user sees for a concept in the progression will be straightforward, whereas later puzzles will be more challenging. I believe that by varying the presentation of the puzzles based on the user's prior knowledge, novice programmer's will be able to acquire new programming skills found within unfamiliar code.

## References

[1] J. Brandt, P. J. Guo, J. Lewenstein, M. Dontcheva, and S. R. Klemmer, "Two Studies of Opportunistic Programming: Interleaving Web Foraging, Learning, and Writing Code," in *Proc. SIGCHI Conference on Human Factors in Computing Systems*, ACM, 2009.

[2] M. B. Rosson and J. M. Carroll, "The Reuse of Uses in Smalltalk Programming," *ACM Trans. Comput.-Hum. Interact.*, Sep. 1996.

[3] M. Rosson, J. Ballin, and H. Nash, "Everyday Programming: Challenges and Opportunities for Informal Web Development," in *Proc. VL/HCC*, IEEE, 2004.

[4] K. J. Harms, D. Cosgrove, S. Gray, and C. Kelleher, "Automatically Generating Tutorials to Enable Middle School Children to Learn Programming Independently," in *Proc. 12th IDC*, ACM, 2013.

[5] K. Harms, "Applying cognitive load theory to generate effective programming tutorials," in *Proc. Visual Languages and Human-Centric Computing (VL/HCC)*, IEEE, 2013.

[6] J. Sweller, P. Ayres, and S. Kalyuga, *Cognitive Load Theory*. Springer, Apr. 7, 2011.

[7] S. Kalyuga and J. Sweller, "Measuring Knowledge to Optimize Cognitive Load Factors During Instruction," *Journal of Educational Psychology*, 2004.

[8] S. Kalyuga, "Rapid Assessment of Learners' Proficiency: A cognitive load approach," *Educational Psychology*, vol. 26, no. 6, 2006.

[9] M. Yudelson and P. Brusilovsky, "NavEx: Providing Navigation Support for Adaptive Browsing of Annotated Code Examples," in *Proc. 12th International Conference on Artificial Intelligence in Education (AIED)*, IOS Press, 2005.

[10] R. Holmes, R. J. Walker, and G. C. Murphy, "Strathcona Example Recommendation Tool," in *Proc. 10th European Software Engineering Conference*, ACM, 2005.