Understanding and Advancing How Programmers Learn from, Design With, and Reuse Online Programming Examples

Andrew Head, PhD Student, UC Berkeley

When programmers use online code examples, they make mistakes. A comprehensive, high-quality body of online documentation has grown in the last decade [10, 9]. This has made it easier than ever for programmers to consult the web to find examples of unfamiliar languages and APIs [3]. While such resources seem like a boon in all regards, observation of programmers reusing examples has revealed a practice fraught with hazards. When programmers learn the bare minimum to do a coding task, they write shoddy code to just barely satisfy requirements, reimplement functionality that stable libraries already support, and spend more time debugging copied code [2, 3].

Code examples have been identified by many CS education researchers as a cornerstone of CS education (e.g., [8]). And while a strong body of literature offers best practices for curating examples for the classroom, this work overlooks an important group: upperclass CS students engaging in formative open-ended projects, and freshly minted software engineers. These groups succeed and fail by the use of in-the-wild programming examples.

My proposed research seeks to **understand the affordances and pitfalls of programmers working with online code examples**. In addition, I aim to develop and study interfaces to support programmers in effectively learning from and using online examples.

To gain a thorough understanding of how online example usage impacts a programmer's learning, design ability, and code quality, I want to answer the following questions for the academic community: $Q1$. With the ease of copying and pasting, how often do programmers actually learn new concepts from code examples? $Q2$. How often can they design new code from the components used? $Q3$. How frequently do they reuse code that contains violations of idioms or undesirable side-effects? $Q4$. What interventions can we envision to enhance programmers' ability to build mental models of the code they find? $Q5$. What interventions can support more successful reuse and modification of found code?

I will pursue $Q1$–$Q3$ by applying standard observational and experimental techniques from HCI. I will conduct controlled lab studies that ask programmers to perform code reuse and modification tasks, with participants sampled from UC Berkeley EECS upperclass students. This is a method I practiced in past work [5].

$Q1$ and $Q2$ must be answered in a necessarily careful way—participants must be encouraged to program in a 'typical' way that makes use of arbitrary online examples. Programmers provided with single hand-picked examples would be unlikely to inspect and reuse code as they would during normal programming activity. So, programmers must be provided with non-trivial, open-ended programming problems (e.g., building a chat server as in [3]). To assess programmers' learning and ability to build new solutions with components they reused, a second experimenter will remotely watch the programming activity. They will formulate questions about randomly sampled APIs a programmer uses during the study session.

$Q3$ requires access to copied-and-pasted code from realistic scenarios. The experiments described above will be a first source of this data. Further data will be obtained by conducting contextual inquiries with student teams from upper division courses, collecting paste data with an apparatus similar to the one described in [7].

On top of this novel knowledge, I will explore the design space of interventions to better support learning, design, and high-quality integration of code found in online examples ($Q4$–$Q5$). The interfaces will match the work habits observed for $Q1$–$Q3$.

While the form the interfaces will take will follow insights from the $Q1$–$Q3$, I propose some intuitions for what might be helpful to programmers. First, automatic, concise, and in-situ code explanations may reduce the cost of finding the information needed to build a solid mental model of what code does. Infrastructure I built in past work [5] already provides a solid technical grounding and design patterns for such explanations. Second, interactions to foster close inspection of code prior to pasting may encourage systematic code inspection for programmers who lack training (e.g., novel in-browser note-taking methods [1]).

This project will yield insights into the learning, design, and programming challenges of programmers working with online code examples, and concrete software systems and techniques for explaining and encouraging engagement with this code. This knowledge and these tools are critical to preparing the next generation of programmers to thrive in industry, for whom learning new skills "on the job" is a core competency [4]. In this way, my work promotes the NSF broader impacts of **increasing and improving public engagement with science and technology**, and **improving STEM education**.

This research will contribute to the academic body of knowledge on end-user software engineering challenges and strategies, and programming tools to support modern paradigms of development. Beyond the research that inspires the current work (e.g., [3, 6]), the research I propose will address our community's need to understand the implications of code reuse of online examples on programmers' ability to develop mental models, design, and develop quality code. I will disseminate our new knowledge of the affordances and pitfalls of online examples among the academic communities of CHI, VL/HCC, and CSE, and publish insights on the design and usability of the software artifacts at UIST. All software developed through this research will be made open source.

[1] A. Bauer and K. R. Koedinger. "Selection-based Note-taking Applications". In: *CHI '07.*

[2] J. Brandt et al. "Opportunistic Programming: How Rapid Ideation and Prototyping Occur in Practice". In: *Proceedings of the 4th International Workshop on End-user Software Engineering.* 2008.

[3] J. Brandt et al. "Two Studies of Opportunistic Programming: Interleaving Web Foraging, Learning, and Writing Code". In: *CHI '09.* 2009.

[4] M. Exter and N. Turnage. "Exploring Experienced Professionals Reflections on Computing Education". In: *Trans. Comput. Educ.* 3 (July 2012).

[5] A. Head et al. "Tutorons: Generating Context-Relevant, On-Demand Explanations and Demonstrations of Online Code". In: *VL/HCC '15.*

[6] M. Ichinco and C. Kelleher. "Exploring Novice Programmer Example Use". In: *VL/HCC '15.*

[7] M. Kim et al. "An ethnographic study of copy and paste programming practices in OOPL". In: *ISESE '04.* Aug. 2004.

[8] E. Lahtinen, K. Ala-Mutka, and H.-M. Jrvinen. "A Study of the Difficulties of Novice Programmers". In: *ITiCSE '05.*

[9] L. Mamykina et al. "Design lessons from the fastest q&a site in the west". In: *CHI '11.*

[10] C. Parnin et al. "Crowd documentation: Exploring the coverage and the dynamics of API discussions on Stack Overflow". In: *Georgia Institute of Technology, Tech. Rep* (2012).