Reading Before Writing: Can Students Read and Understand Code and Documentation?

Tammy VanDeGrift

February 2005, (SIGCSE)

1 Introduction

Before diving into a description of this proposed research study, I must warn you that these ideas are very premature. (I have spent the bulk of my time writing my dissertation and looking for jobs.) This research study is motivated by an idea I have for teaching introductory computer science. When children learn how to write (more than just the alphanumeric characters), they have spent several years of reading and speaking the language. Therefore, when they are learning to write, children have an existing framework governing syntax and semantics of the language. Also, by spending several years reading, children are exposed to several models of writing before they write something themselves. My primary interest for this study is finding out if students can read and understand programs. A second motivation for my interest in teaching students to read code before writing code also comes from professional industry. In casual conversations with software developers I have asked what skills are they looking for in computer science graduates. I often get the standard responses, such as "they can think" and "they can actually write code". Another response is that they can understand existing code since most developers maintain and extend existing systems. My overall research question is not something that I cannot investigate immediately. First, I would like to do preliminary studies focusing on more direct questions. My overall research question is the following: does teaching students how to program by first reading programs lead to more efficient learning of language syntax and semantics? First, I will focus on the question of can students currently read and understand existing code and documentation.

2 Related Work

This study draws on the reading/writing literature and the code comprehension literature. Musthafa documents the history of research practices in reading and writing from the 1960's through the 1990's in [5]. Reading and writing were first viewed as simply behavioral responses and that reading and writing were separate acts. Later, reading and writing theories about causal relationships were studied. Does reading lead to writing or does writing lead to reading? Most recently, the trend is that reading and writing are related activities and each may help the other. It is my sense that when teaching students how to program, there could be a great synergy between writing programs and reading programs to aid in becoming better at both reading and writing. A popular approach to teaching reading is to have kids read and write about the reading [1]. Cobine argues that encouraging kids to write about their reading will satisfy a larger set of learning styles. In the same way, teaching programming through reading and writing may reach a more diverse set of learning styles. The second body of literature related to my interests is the program comprehension, program understanding, debugging, and expert/novice literature. Shaft uses a think-aloud protocol to collect data about how people comprehend a program unfamiliar to them [7]. She chose participants who had several years of industry experience with COBOL, so her sample population is different than my target population. She focused on programmers' use of metacognition while reading programs. She found that when they were working in unfamiliar application domains, the use of metacognition interfered with their comprehension skills. Mayer addresses effective techniques for teaching students how to program, such as providing models and having students describe a concept in their own words, in [4]. Missing from this

paper is the technique of having students read programs before or while learning how to write them. Several researchers have investigated expert/novice differences in program comprehension, theories about program comprehension, and tools to assist in comprehension [6, 3, 2]. One day, there might be tools to assist novices in learning how to read code, but we need to first understand how they read and understand code without assistance and the challenges they face when reading code.

3 Research Questions

My research question is: Can students read and understand existing code and documentation? Embedded in this question is: How do students read and understand existing code and documentation? Can students evaluate if existing code meets the functional requirements? Can students predict the output of a program given an input or event? Can students follow the execution path of a program given an input or event?

(Okay, so maybe I have not quite focused the question enough to be able to develop a study to investigate it.)

4 Evidence

Evidence to the main question of students being able to read and understand existing code and documentation can take on the following forms:

- A student asks a question about the code. This shows that the student knows at least something about the code to ask about it.
- A student can tell me if the code does or does need meet a functional requirement.
- A student can tell me that the code documentation is inconsistent with the code.
- A student can predict the outcome of the program given a certain input.
- A student can trace the execution path given a certain input.
- A student can find an unhandled case in the code.
- A student looks up a language feature in the reference manual. This shows that the student knows that they are not familiar with a language feature, instead of blindly skipping over that part of the code.
- A student draws pictures showing connections between files/classes/code fragments or abstracts the code into a diagram.

5 Methods

Looking at my questions again, I think I would split the investigation into two phases. Phase one will look at can students read and understand code and phase two will center around how do students read and understand code. It might be feasible to study both questions at the same time or with two separate phases in a single experiment. Since I do not know if students can read and understand code, I would probably recruit participants from across all levels in an undergraduate CS program. The most novice set of participants would be students who have just completed CS 1.

I would take a more experimental approach when studying this question, instead of incorporating it into an actual class. I would design the experiment so participants were asked to do the following tasks:

1. After introducing the study, I would let the participants know that they can ask me questions at any time. Participants would first be given the code (not sure if this will be done on paper or on the computer), access to a language reference, some description/documentation about what the code does, and a set of functional requirements for the code. The participant would be given sufficient time to read through the material and I would ask him/her to read it and I would be asking questions about the program during the next phase.

2. After the participant is comfortable with the materials, I would ask him/her to describe the program as if he/she is introducing the system to a peer.

3. The participant would then be asked a set of questions about the program. I would ask what would happen given a certain input. I would ask the participant to trace the execution path given a certain input. I would also ask the participant to confirm if the program satisfies a certain functional requirement.

4. The second phase is for the investigation of the question as to how students read and understand code. (the process in which they do this) I would give the participant a second program and documentation and ask them to think aloud - tell me what they are looking at and what they are thinking as they try to understand the code. This could be tricky, since having them think aloud may alter the process through which they go through. (Having two passes at this gives me some evidence about the thinking aloud altering their process. While they read through the first program, I would be taking notes as to what they are looking at, when they look at the manual, when they ask me questions, etc.). An alternative to using a second program would be to videotape the participant while he/she was trying to understand program 1. They I could ask him/her what he/she was doing/thinking while watching the tape.

5. I would then ask the participant to describe how they try to understand the code for a system that is unfamiliar to them. I would ask them to describe any differences in the process they went through for the two programs.

6 Analysis Procedures

I expect to analyze the data both quantitatively and qualitatively. The data I expect to get from the experiment include the following:

- Descriptions of process when reading/understanding program 1 (from my observations)
- Descriptions of process when reading/understanding program 1 from participant's answer about their process
- Questions that the participant asks during the experiment
- The answers to the input/output questions
- Participant's description of program 1
- The answers to the questions about tracing execution
- The answers to the questions about the program meeting functional requirements
- The verbatim transcript from the think aloud protocol for program 2
- Descriptions of differences for the participant in reading program 1 and program 2
- Descriptions of how the participant usually goes about understanding the code for an unfamiliar system

With the descriptions of process, I would try to build timelines of what the students were doing and the order in which they did them. I would compare their descriptions to the one I observed. I would see how many questions they got right about the input/output, tracing, and meeting functional requirements. I would also look at their verbal description of the program, annotating correct and incorrect interpretations of the program.

7 Future Studies

My general interest is to find out if teaching students to read before writing programs is more efficient for their learning. In the future I could try to incorporate code reading exercises at the beginning of the term and see how quickly students can get through the syntax and semantics of the language. If students can learn about the language more quickly, then class time could be spent on issues such as efficiency, design decisions, and readability of software.

References

[1] Gary R. Cobine. ED386734 1995-00-00 Writing as a Response to Reading. *ERIC Digest*, 1995.

[2] Vikki Fix, Susan Wiedenbeck, and Jean Scholtz. Mental Representations of Programs by Novices and Experts. In *Proceedings of INTERCHI*, pages 74–79, 1993.

[3] Diane Kelly and Terry Shepard. Qualitative Observations from Software Code Inspection Experiments. In *Proceedings of the 2002 Conference of the Center for Advanced Studies on Collaborative Research*, 2003.

[4] Richard E. Mayer. The Psychology of How Novices Learn Computer Programming. *Computing Surveys*, 12(1):121–141, March 1981.

[5] Bachrudin Musthafa. Reading-Writing Connections: Shifts in Research Foci and Instructional Practices. *Education Resources Information Center*, 1996. ED 396 275.

[6] Santanu Paul, Atul Prakash, Erich Buss, and John Henshaw. Theories and Techniques of Program Understanding. In *Proceedings of the 1991 Conference of the Center for Advanced Studies on Collaborative Research*, pages 37–53, 1991.

[7] Teresa M. Shaft. Helping Programmers Understand Computer Programs: The Use of Metacognition. *Data Base Advances*, 26(4):25–46, 1995.