Leveraging Analyses of Big Data on Computing Students’ Programming and Social Interaction Processes to Build More Effective Programming Environments and Pedagogical Interventions

Christopher D. Hundhausen
Human-centered Environments for Learning and Programming (HELP) Lab
School of Electrical Engineering and Computer Science
Washington State University
Pullman, WA 99164-2752
hundhaus@wsu.edu

Introduction
Especially in early computing courses, students spend a majority of their time solving programming problems outside of class. How much do we really know about what they do as they solve programming problems using an integrated development environment (IDE)? While some small-scale empirical studies have appeared in the computing education literature [see, e.g., Hundhausen et al. 2006], the study of students’ programming processes remains in its infancy. Within the last decade, researchers have begun to utilize software logging as a means of capturing students’ programming processes. Jadud [2006], for example, analyzed software logs of introductory computing students’ compilation behaviors, and developed a behavioral metric called the error quotient that was shown to be strongly correlated with student learning outcomes.

Further insights into learners’ programming processes—including their edits, compilation attempts, run-time exceptions, and debugging actions—hold promise in helping us to improve the design of IDEs and instructional methods for computing. However, analyzing these events proves difficult, as programming is such a contextual activity. It’s one thing to observe that a given learner is making a lot of error-free compilations, and that the learner’s execution of the program does not result in run-time exceptions. It’s quite another thing to understand what resources the learner brought to bear on that activity, and whether the run-time behavior is actually correct.

The advent of social networking makes it possible gain further insights into learners’ programming activities. In a research project recently funded by the National Science Foundation’s Cyberlearning program (no. IIS-1321045), my colleagues and I are exploring the educational affordances of integrating a social networking-style activity stream directly into an IDE. Thus, learners are able to pose questions about, and identify issues with, their programming activities in the same place where those questions and issues arise. They are even able to ground their questions and issues firmly in the programming artifacts (code, call stack, variable watch window) to which they refer.

Aside from providing students with a technology-rich environment in which to learn computer programming, our proposed “social programming environment” opens up new opportunities to study learners’ situated programming practices. Indeed, critical contextual details of learners’ programming practices, revealed through asynchronous discussions about those practices, can help us to better understand their intentions, struggles, and strategies. With respect to the analysis of “big” data on students’ programming processes, possibly including their situated asynchronous conversations about programming, several intriguing research questions arise:

RQ1: What problems and issues do learners encounter as they learn programming? Along what dimensions can these problems and issues be fruitfully classified?

RQ2: What resources and solution strategies do learners employ in order to address their problems and issues? To what extent do these resources and solution strategies generalize across problem solving situations, and to what extent do resources and solution strategies depend upon specific problems and learners?
RQ3: Are certain strategies, resources, and learner characteristics correlated with higher and more efficient problem solving success?

All of these questions lead to the following “grand challenge” question

RQ4: How can an understanding of learners’ usage of resources and strategies while solving computing problems be leveraged to build more effective programming environments and pedagogical interventions?

In the remainder of this white paper, I sketch out a possible research program for exploring these questions—one that is already underway at Washington State University [Hundhausen and Carter 2013]. I then consider the potential impact of this research on computing education.

Research Program

In order to be in a position to study students’ programming processes and social interactions in detail, one first needs to build a prototype programming environment that not only performs event logging, but also supports the ability to socialize with others about programming activities. To that end, my Ph.D. student Adam Carter and I have developed a “social plug-in” for Microsoft Visual Studio, a commercial IDE (see Figure 1). The plug-in supports an activity stream (right-hand side of Figure 1) that presents a learning community’s programming activities as they happen. As these activities—including compilations, run-time exceptions, and debugging events—unfold, a back-end social recommender relates them to other learners’ activities (e.g., “You and 3 others have gotten this error”). In addition, learners can search and filter the activity stream for events that might help them with their programming problems and issues. If no helpful events can be found, learners always have the option of selecting programming artifacts—sections of code, pieces of the call stack, sections of variable watch windows, and the like—and then selecting “Ask a question...” from a pop-up context menu. A snapshot of the relevant programming artifact is then injected into the activity stream, making it easy for the learner to ask a question about it. Finally, learners can pose and respond to questions directly by commenting on any item in the activity stream, just as they can comment on posts on social networking sites such as Facebook.

Figure 1. Microsoft Visual Studio with Social Plug-in
There are obviously a number of important research questions surrounding the design of such a “social programming plug-in.” We are exploring these questions as part of a research project entitled “Exploring Social Programming Environments in Early Computing Courses,” which was recently funded by the NSF Cyberlearning Program (no. IIS-1321045). In this white paper, however, the focus is not on the design of the environment, but on the use of the environment as part of a “big data” research project that explores learners’ programming activities. To that end, we have built an event logging system into Visual Studio. Our event logging system is similar to other efforts in the field, including Jadud’s initial logging system for BlueJ [Jadud 2006], as well as the more recent effort in the BlueJ community to collect log data [Utting et al. 2012]. As they work in Visual Studio, learner’s events are sent to an event repository, which stores the events in a database. This database, which contains both programming and social events, forms the basis for the proposed “big data” research program. The idea is to collect IDE log data not only from our own “social programming environment,” but also from other IDEs such as BlueJ, and to analyze the corpus of log data as a whole relative to the research questions posed above. In the case of log data obtained from our own “social programming environment,” we will be able to triangulate programming event data with social event data in order to obtain a more contextualized picture of learners’ struggles and strategies.

Will an analysis of log data alone provide sufficient insights into the research questions we have posed? While I conjecture that such an analysis will provide an excellent starting point, I believe that additional data collection efforts will be needed to address the questions in a more nuanced way. For example, in order to correlate learner activities with individual differences, we will need to collect background survey data, including age, gender, computer programming experience, grade point average, and attitudinal measures that prior research has shown to predict computing success (e.g., the computer playfulness scale [Webster and Martocchio 1992]). Likewise, in order to correlate learner activities with task success and learning, we will need to collect data on programming assignment grades, administer standardized pre- and post-tests of computing knowledge [Tew and Guzdial 2011], and administer individual closed-condition programming post-tests. Finally, in order to correlate student programming activities with relevant attitudinal measures, we will need to administer pre- and post-surveys using instruments such as the Motivated Strategies for Learning Questionnaire [Pintrich et al. 1991], a programming self-efficacy scale [Ramalingam and Weidenbeck 1998], the classroom community scale [Rovai 2002], and the Computing Attitudes Survey [Tew et al. 2012]. Through a systematic large-scale effort, we could collect these data at many different institutions using different programming languages and IDEs. Alongside the event log data, this complementary source of “big data” would help us to gain further insights into the research questions posed.

Impact on Computing Education

The proposed research program has the potential to make a significant impact on computing education research. First, by leveraging data corpora of unprecedented size, the research holds promise in contributing new, theoretically-grounded accounts of how learning unfolds in IDEs. This is important because, while there have been several attempts to understand and characterize computer science learners’ problem-solving behaviors, these attempts have been based on relatively small samples of data, and have not attempted to consolidate data across multiple languages, IDEs and institutions.

Second, the proposed research program has the potential to serve as an empirical foundation for next-generation IDEs and best practices for computing education. For example, this research opens up new opportunities to develop adaptive IDEs that are able to recognize learner problems and issues as they unfold, and to gently guide learners toward more productive programming practices based on actual evidence of what has worked for learners in similar situations in the past. With respect to shaping best practices, this research opens up new opportunities for instructors to tailor their instruction based upon better understandings of learners’ struggles and problems. Indeed, a logical extension of the research would be to develop an instructor dashboard: a visual analytics environment in which instructors can visually explore their students’ programming activities and progress in aggregate, with an eye toward tailoring their instruction and feedback to address the current needs of their students. For example, suppose, through a visual analytics environment, an instructor discovers that students are frequently encountering a similar runtime exception. In response, the instructor could focus her instructional efforts on addressing the underlying misconceptions that lead to the buggy code that is generating the exception.

The grand vision of this line of research is learn as much about learners’ programming practices as Google knows about consumers. Just as Google aims to understand consumers’ behaviors so that it can best tailor advertisements and services, so too could we, as computing education researchers, aim to understand learners’ programming behaviors so that we can best tailor our programming technologies and pedagogical practices.
References


