Students as designers of programs: Why we should and how we can study the practices and artifacts that lead to students’ final code

In what follows, I do three things.

1. I outline a gap in existing Computer Science Education (hereafter CSEd) research: we lack a detailed understanding of how students design software. With a few notable exceptions (for example, [3]–[6]), most research does not view students as designers. Consequently, there is a paucity of research on how students understand the task of software design, what guides their moment-by-moment activity as they work on a design, and how the final form of code reflects their design knowledge. Emerging research shows what professional software design looks like in practice, but that only fleshes out the “expert” end of a novice-expert continuum. We’re still left with an asymmetrical model of design expertise: we know far more about the target than we do the origin or the paths between.

2. I propose how we can study students — specifically high school and university students — as designers. Studying students as designers means looking not just at final code submissions but at the interactions that happen between compilations and in early stage design, sometimes before a single line of source code is ever written.

3. I explain why pursuing this research matters. I underscore the importance of treating students as designers and show how such an orientation helps us make progress in both theory-building and instructional reform.

1. Studying students’ design practices helps flesh out our sense of the novice-expert continuum for software design

In 2010, the journal Design Studies devoted an entire issue to how professional software engineers design complex systems [2]. That issue’s editors argued fields of design studies, interaction analysis, and human-computer interaction don’t know enough about how software engineers use representations and collaborative exchanges to organize the beginning phases of a design:

During formative design, software engineers spend a great deal of time engaging in creative, exploratory design thinking using pen and paper or a whiteboard—whether alone or in a small group. However, not enough is known about how software designers work in such settings. What do designers actually do during early software design? How do they communicate? What sorts of drawings do they create? What kinds of strategies do they apply in exploring the vast space of possible designs? [2, p. 533]; my emphasis added

And, while these questions were reprised in a 2012 issue of IEEE Software [1], we are still in the early stages of answering them.

Careful study of how experts design software shows the complexity involved in producing code [7]–[9]. Such results echo findings from extensive studies of engineers and scientists designing in practice [10]–[15]. It’s striking, then, that there is no body of research exploring how students engage in design when they code. Instead, the latest generation of research on student learning in programming has focused much more extensively on questions like how can we assess and mitigate
students’ difficulty in programming than it has on questions like how do students learn and display evidence of design thinking in programming?

That difference is subtle, but it bears repeating. If we rephrase the quoted questions [2, p. 553] above and treat students as designers, we find the following questions that I think should drive a program of CSEd Research:

• What does students’ exploratory design thinking look like?
  ○ What do students actually do during early stage design work?
  ○ How do students communicate?
  ○ What sorts of drawings do students create?
  ○ What kinds of strategies do students apply in exploring the vast space of possible designs?

I think such research is both possible and potentially fruitful. We can do it using methods that already exist. And, it stands to greatly inform how we theorize students’ capacity for design in programming.

2. A program of study should analyze the inscriptive, gestural, verbal, and computational artifacts students create in design

The methods below form the core of my proposed program to study students as software designers. None of the methods below are new; all have been used in prior educational research. What is new, I believe, is the opportunity to combine them all under the umbrella of understanding what happens when students design software.

• We should collect students’ code history. Research has already shown code snapshotting to be a useful method for understanding large-scale patterns of student error [16]–[18]. But, that research takes an aggregate view. An untapped advantage of collecting code history data is that it gives us a fine-grained record of how individual students’ designs evolve.

• We should conduct clinical interviews with students. Clinical interviews have proven historically useful in understanding the substance and structure of participant’s knowledge [19]–[23]. Crucially for CSEd Research, clinical interviews can tell us how students view knowledge and knowing in a discipline [24] which can affect how they approach and adopt that discipline’s practices [23], [25], [26].

• We should analyze students’ in-interview inscriptions — what they write, how they write it, and how it gets used. Evidence from both science studies [12]–[14], [27], [28] and educational research [15], [29], [30] highlights the centrality of inscriptions in disciplinary practice. Software design is no exception [8], [31].

• We should analyze students’ in-interview gestures. Perspectives of gestural analysis hold that gestures can not only support or extend thinking, they can also communicate entirely new information [32]. Moreover, perspectives on embodied cognition argue that bodily motion is itself cognition [33], [34]. Whether or not one subscribes to the strongest tenets of embodied cognition, it still urges us to uncover the role of the physical environment in students’ design thinking.
Figure 1 (below) shows a schematic overview of these methods in action: a voice recorder for speech, LiveScribe pen for inscriptions, camera for gestures, and an in-interview computer tracking code history and capturing real-time activity.

Figure 1 – An example of coordinating multiple modes of data and analysis to understand a student’s design process

3. Early research suggests this paradigm can help us build better theories of learning and improve instructional interventions

As an example of what this research orientation can provide, my dissertation work explored whether and how students use pseudocode in their design work. I found both the form pseudocode takes and the uses students put it to vary widely. One student, Lionel, writes pseudocode as an absolutely essential part of convincing himself he understands the top-level view of a program. Another student, Rebecca, uses it sparingly, and even then only as a place-holder when she feels like she’s not sure how to actually write a procedure properly in the language. Even if Rebecca and Lionel both produce working problems as a design product, their trajectories look markedly different. Moreover, an instructor’s field of options in helping each student varies, not because of conceptual knowledge they have or don’t have, but because of differing design practices they’ve internalized.

Ultimately, attending to students’ design processes opens up new possibilities for research and practice. Sensitive research can help us understand students’ productive capacities for design. As argued above, we need to understand the knowledge and experiences students already have for design if we aim to model the path toward expertise using constructivism. In a parallel way responsive, informed instruction depends on a deep understanding what students think they’re doing when they’re working on programming projects.
Danielak


