Studying Students' Early-Stage Software Design Practices

Brian A. Danielak, University of Wisconsin-Madison, danielak@wisc.edu
William E. J. Doane, Institute for Defense Analyses, wdoane@ida.org

Abstract: This paper describes research on undergraduate engineering students designing computer programs. We do three things. First, we explain what early-stage software design practices are and argue why we should look for students' productive capacities in them. Second, we outline the basics of a developing research program that aims at carefully documenting students' capacities for doing software design. Third, we relate early results from that research.

Introduction and Overview

In what follows, we do three things.

1. We 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 (Harel & Papert, 1992; Papert, 1980; Peppler & Kafai, 2008; Turkle & Papert, 1990), most CSEd 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 how software design expertise—as opposed to simply programming expertise, debugging practices, etc.—develops: we know far more about the target than we do possible origins or the paths between. That problem carries over to instruction. To answer questions of what students *should* do as a practice in software design (cf., Shaffer & Resnick, 1999), we need to first explore what they *can* do.
2. We propose *how* researchers can study students—specifically high school and university students—as *designers* of computer programs. Studying students as designers means looking not just at final code submissions but at what happens *between* compilations. It also means looking at students' early-stage activity: the interactions and artifacts that can emerge before a single line of source code is ever written.
3. We explain *why* pursuing this research matters. We 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.

We Lack Accounts of Students' Software Design Practices

In 2010, the journal *Design Studies* devoted an entire issue to how professional software engineers design complex systems (Petre, van der Hoek, & Baker, 2010). 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? (Petre et al., 2010, p. 533)

These questions were reprised in a 2012 issue of *IEEE Software* (Baker, van der Hoek, Ossher, & Petre, 2012), but we are still in the early stages of answering them.

Careful study of how experts design software is starting to reveal the complex disciplinary practices that comprise early-stage work (Ball, Onarheim, & Christensen, 2010; Jackson, 2010; Rooksby, 2010). For example, Rooksby and Ikeya (Rooksby & Ikeya, 2012; Rooksby, 2010) analyzed video recordings of three professional software engineer teams in the early stages of a software design task. Their research reveals that whiteboards—common artifacts in software design work—serve a much more complex purpose than simply to "log ideas and decisions" (Rooksby & Ikeya, 2012, p. 58). Moreover, the inscriptional content of the whiteboard is only a small part of design-work's complexity. Also crucial are the practices—such as conveying *epistemic uncertainty* about design solutions (Ball et al., 2010)—that help designers "remain coordinated and focused while collaborating" (Rooksby & Ikeya, 2012, p. 56). Ultimately, students need to learn as much about these

practices "as they do about procedures and technologies" when learning software design (Rooksby & Ikeya, 2012, p. 60)

The above results from software engineering echo findings from deep studies of engineers and scientists working and designing in practice. Practicing professionals make complex use of talk, gesture, and representational artifacts in physics (Gupta, Hammer, & Redish, 2010; Kaiser, 2005; Ochs, Gonzales, & Jacoby, 1996); chemistry (Stieff, 2007); field biology (Hall, Stevens, & Torralba, 2002); civil engineering (Stevens & Hall, 1998); structural engineering (Gainsburg, 2006); mechanical and electrical engineering (Bucciarelli, 1994; Henderson, 1999); and architectural design (Hall et al., 2002). Given that:

1. Science and engineering education research has made progress by looking for continuities in how novice learners develop disciplinary practices (Gupta et al., 2010; Hall & Stevens, 1995; Stevens & Hall, 1998), and
2. Emerging research on software engineering reveals that early-stage software design involves complex inscriptional, discursively, and epistemic practices,

it seems striking that there is no contemporary body of research, comparable to studies of expert practice, that looks at students' software design practices. In other words, we have every reason to believe that expertise in software design involves complex practices, but little (if any) research on what productive capacities students have for them. Finding those productive design capacities requires a shift from questions such as *how can we assess and mitigate students' difficulty in programming?* toward questions such as *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 above (Petre et al., 2010, p. 533) and treat *students* as designers, we find the following questions that we think can drive a program of CSEd Research:

- What do students actually do during early stage software design work?
- What does students' exploratory design thinking look like?
- How do students communicate?
- What sorts of drawings do students create when they design software?
- What kinds of strategies do students apply in exploring the vast space of possible software designs?

We 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 build theory about students' capacities for design in programming.

Studying Students' Design Practices Means Analyzing the Inscriptional, Gestural, Discursive, and Computational Artifacts They Create When Building Programs

The methods below form the core of our developing 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, we 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*. By this, we mean we should preserve frequent snapshots in time of what students' code looks like. Research has already shown code snapshotting to be a useful method for understanding large-scale patterns of student error (Jadud, 2006; Rodrigo, Tabanao, Lahoz, & Jadud, 2009; Spacco et al., 2006). And, the resolution of snapshots is extremely fine: Spacco et al. are able to capture the contents of a file each and every time a student saves it to disk. But, that research takes an aggregate view: it identifies large-scale error patterns at the expense of detailed naturalistic understandings of why students make those errors. Moreover, it's primarily used to identify what *mistakes* students make, which is distinctly different from a research orientation that considers the negative *and* positive consequences of students' design choices. A currently untapped advantage of collecting code history data, then, is that while we historically use it to aggregate *across programmers* it nevertheless also gives us fine-grained individual or team-based histories of how designs evolve.

We should conduct *clinical interviews* with students. Clinical interviews have proven historically useful in understanding the substance of students' knowledge and the nature of conceptual change (diSessa & Sherin, 1998; Duckworth, 2006; Ginsburg & Opper, 1988; Ginsburg, 1997). Crucially for CSEd Research, clinical interviews can tell us about students' epistemologies—how they view knowledge and knowing in a discipline (Hofer & Pintrich, 1997)—which can affect how they approach and adopt that discipline's practices (Hammer, 1989, 1994; Lising & Elby, 2005). Moreover, when interviews are videotaped and analyzed from perspectives of interaction analysis (Goodwin, 2000; Jordan & Henderson, 1995), interviews can offer rich evidence of the substance of students' design practices

We should analyze students' in-interview *inscriptions* when they design — what they write, how they write it, and how those writings get used. Evidence from both science studies (Hall et al., 2002; Henderson, 1999; Hutchins, 1995; Kaiser, 2005; Latour, 1990; Ochs et al., 1996) and educational research (Hall & Stevens, 1995; Lehrer, Schauble, Carpenter, & Penner, 2000; Stevens & Hall, 1998) highlights the centrality of

inscriptions to disciplinary practice in science and mathematics. Ethnomethdological data from studies of professional software engineers supports the same result: inscriptional practice is central to how professional engineers design software (Rooksby & Ikeya, 2012; Rooksby, 2010). Because part of that inscriptional environment is the computer itself, we should strive to capture and analyze what happens on-screen as students design programs.

We should pay close attention to students' in-interview *gestures*. Perspectives of gestural analysis hold that gestures can not only support or extend thinking, they can also communicate entirely different information than what's being said (Goldin-Meadow, 2003). Moreover, perspectives from cognitive anthropology and embodied cognition studies argue that bodily motion *is itself* cognition (Hall & Nemirovsky, 2012; Hutchins, 1995; Nemirovsky, Rasmussen, Sweeney, & Wawro, 2012). For example, when students describe a part of code by moving their hand across an imaginary row of items and tapping each item, their bodies convey information we can interpret about how they understand iteration. 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' software design thinking.

Figure 1 presents a visual overview of some of these methods and modes of analysis. In particular, the first (top-left) panel depicts how we deploy these data collection methods during a clinical interview:

- a voice recorder captures speech (often a redundancy in case another recording device fails)
- a LiveScribe Pulse pen digitizes written inscriptions, allowing us to play back what was written in time
- a videocamera records data for knowledge analysis, interaction analysis, and gestural analysis,
- an in-interview computer tracks code history and screen-capturing software records real-time activity.

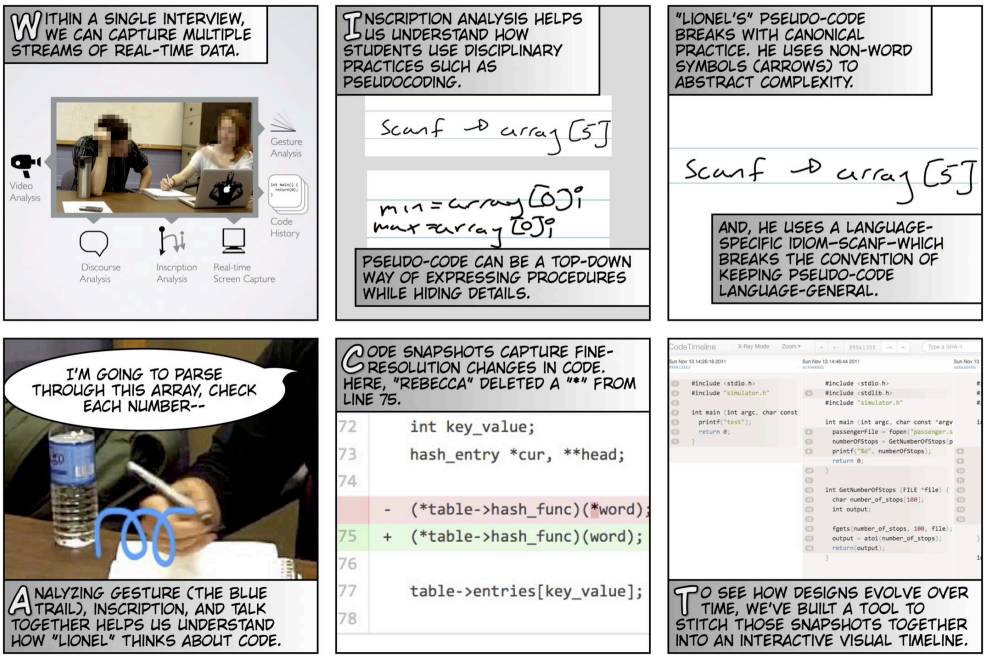

Figure 1. An overview of select strategies for capturing students' design practices.

Early Results Suggest Students View and Enact Disciplinary Practices with Remarkable Diversity

This paper reflects only one aspect of a larger study reported in Danielak (2014) to motivate a conversation between the learning sciences community and the computer science education research community. As an example of what this research orientation can provide, one strand of our research asks whether and how students use pseudocode in their design work. Pseudo-code is a high-level, abstract way of expressing procedures while backgrounding the details of how those procedures are carried out. It is also meant to be language-neutral, such that procedures expressed in pseudocode should be implementable in a variety of specific programming languages. We found that both the form pseudocode takes and the uses students put it to vary widely. One student, "Lionel," writes pseudocode as a prospective design tool. In Lionel's design process, writing pseudocode before writing language-specific program code is an absolutely essential part of convincing himself he understands the top-level view of what his program will do. Moreover, Lionel makes opportunistic use of language-specific idioms. Using such language-specific programming constructs would seem to clash with professional guidelines for writing pseudocode. Nevertheless, Lionel makes unproblematic use of his hybrid

syntax pseudocode to successfully solve problems. Another student, "Rebecca," uses pseudocode sparingly. Rebecca writes pseudocode when she feels unsure of how to write a procedure properly in the language she's using. For Rebecca, pseudocode becomes a fallback; a place-holder for the code she thinks belongs in a section of her program, but does not currently know how to write. Crucially, our data suggest when Rebecca uses pseudocode it seems to coincide with her feeling diffident about her programming ability. For Rebecca, pseudocode becomes a concession.

The contrast between how Lionel and Rebecca write and view pseudocode is striking. While Lionel uses pseudocode deliberately to plan and understand, Rebecca uses it hesitantly, viewing it as non-working code she must ultimately replace with working code. Even if Rebecca and Lionel both produce working programs 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 students 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 formatively understand students' productive capacities for design. We can conduct more powerful studies of design knowledge *in transition* (Smith, diSessa, & Roschelle, 1993). As argued, 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 and informed instruction depends on a deep understanding of what students think they're doing when they're doing software design.

References

Baker, A., van der Hoek, A., Ossher, H., & Petre, M. (2012). Guest Editors' Introduction: Studying Professional Software Design. *Software, IEEE*, *29*(1), 28–33. doi:10.1109/MS.2011.155

Ball, L. J., Onarheim, B., & Christensen, B. T. (2010). Design requirements, epistemic uncertainty and solution development strategies in software design. *Design Studies*, *31*(6), 567–589. doi:10.1016/j.destud.2010.09.003

Bucciarelli, L. L. (1994). *Designing engineers*. Cambridge, Mass: MIT Press.

Danielak, B. A. (2014). *How electrical engineering students design computer programs*. University of Maryland, College Park.

diSessa, A. A., & Sherin, B. L. (1998). What changes in conceptual change? *International Journal of Science Education*, *20*(10), 1155–1191. doi:10.1080/0950069980201002

Duckworth, E. R. (2006). *"The having of wonderful ideas" and other essays on teaching and learning* (3rd ed.). New York: Teachers College Press.

Gainsburg, J. (2006). The mathematical modeling of structural engineers. *Mathematical Thinking & Learning*, *8*(1), 3–36. doi:10.1207/s15327833mtl0801_2

Ginsburg, H. P. (1997). *Entering the child's mind: the clinical interview in psychological research and practice*. Cambridge□; New York: Cambridge University Press.

Ginsburg, H. P., & Opper, S. (1988). *Piaget's Theory of Intellectual Development* (3rd ed.). Englewood Cliffs, N.J: Prentice-Hall. Retrieved from http://lccn.loc.gov/87017353

Goldin-Meadow, S. (2003). *Hearing gesture: how our hands help us think*. Cambridge, Mass: Belknap Press of Harvard University Press.

Goodwin, C. (2000). Action and embodiment within situated human interaction. *Journal of Pragmatics*, *32*(10), 1489–1522. doi:10.1016/S0378-2166(99)00096-X

Gupta, A., Hammer, D., & Redish, E. F. (2010). The Case for Dynamic Models of Learners' Ontologies in Physics. *Journal of the Learning Sciences*, *19*(3), 285. doi:10.1080/10508406.2010.491751

Hall, R., & Nemirovsky, R. (2012). Introduction to the Special Issue: Modalities of Body Engagement in Mathematical Activity and Learning. *Journal of the Learning Sciences*, *21*(2), 207–215. doi:10.1080/10508406.2011.611447

Hall, R., & Stevens, R. (1995). Making space: A comparison of mathematical work in school and professional design practices. In S. L. Star (Ed.), *The cultures of computing* (pp. 118–145). Oxford, UK: Blackwell Publisher.

Hall, R., Stevens, R., & Torralba, T. (2002). Disrupting representational infrastructure in conversations across disciplines. *Mind, Culture & Activity*, *9*(3), 179–210.

Hammer, D. (1989). Two approaches to learning physics. *The Physics Teacher*, *27*(9), 664–670. doi:10.1119/1.2342910

Hammer, D. (1994). Epistemological beliefs in introductory physics. *Cognition and Instruction*, *12*(2), 151–183. doi:10.2307/3233679

Harel, I., & Papert, S. (1992). Software design as a learning environment. *Learning to Design, Designing to Learn: Using Technology to Transform the Curriculum*, 35–70.

Henderson, K. (1999). *On line and on paper: visual representations, visual culture, and computer graphics in design engineering*. Cambridge, Mass: MIT Press.

Hofer, B. K., & Pintrich, P. R. (1997). The development of epistemological theories: Beliefs about knowledge and knowing and their relation to learning. *Review of Educational Research*, *67*(1), 88–140. doi:10.3102/00346543067001088

Hutchins, E. (1995). *Cognition in the Wild*. Cambridge, Mass: MIT Press.

Jackson, M. (2010). Representing structure in a software system design. *Design Studies*, *31*(6), 545–566. doi:10.1016/j.destud.2010.09.002

Jadud, M. C. (2006). Methods and tools for exploring novice compilation behaviour. In *Proceedings of the 2006 international workshop on Computing education research - ICER '06* (p. 73). Canterbury, United Kingdom. doi:10.1145/1151588.1151600

Jordan, B., & Henderson, A. (1995). Interaction analysis: Foundations and practice. *Journal of the Learning Sciences*, *4*(1), 39. doi:10.1207/s15327809jls0401_2

Kaiser, D. (2005). *Drawing theories apart: the dispersion of Feynman diagrams in postwar physics*. Chicago: University of Chicago Press.

Latour, B. (1990). Drawing things together. In M. Lynch & S. Woolgar (Eds.), *Representation in Scientific Practice* (1st MIT Press ed., pp. 19–68). Cambridge, Mass: MIT Press.

Lehrer, R., Schauble, L., Carpenter, S., & Penner, D. (2000). The interrrelated development of inscriptions and conceptual understanding. In P. Cobb, E. Yackel, & K. McClain (Eds.), *Symbolizing and Communicating in Mathematics Classrooms: Perspectives on Discourse, Tools, and Instructional Design* (pp. 325–360). Mahwah, N.J: Lawrence Erlbaum Associates.

Lising, L., & Elby, A. (2005). The impact of epistemology on learning: A case study from introductory physics. *American Journal of Physics*, *73*(4), 372. doi:10.1119/1.1848115

Nemirovsky, R., Rasmussen, C., Sweeney, G., & Wawro, M. (2012). When the Classroom Floor Becomes the Complex Plane: Addition and Multiplication as Ways of Bodily Navigation. *Journal of the Learning Sciences*, *21*(2), 287–323. doi:10.1080/10508406.2011.611445

Ochs, E., Gonzales, P., & Jacoby, S. (1996). "When I come down I'm in the domain state": grammar and graphic representation in the interpretive activity of physicists. In *Interaction and Grammar* (pp. 328–369). Cambridge: Cambridge University Press.

Papert, S. (1980). *Mindstorms: Children, Computers, and Powerful Ideas*. New York: Basic Books. Retrieved from http://lccn.loc.gov/79005200

Peppler, K. A., & Kafai, Y. B. (2008). Youth as media art designers: workshops for creative coding. In *Proceedings of the 7th international conference on Interaction design and children* (pp. 137–140). New York, NY, USA: ACM. doi:10.1145/1463689.1463740

Petre, M., van der Hoek, A., & Baker, A. (2010). Editorial. *Design Studies*, *31*(6), 533–544. doi:10.1016/j.destud.2010.09.001

Rodrigo, M. M. T., Tabanao, E., Lahoz, M. B. ., & Jadud, M. C. (2009). Analyzing Online Protocols to Characterize Novice Java Programmers. *Philippine Journal of Science*, *138*(2), 177–190.

Rooksby, J. (2010). "Just try to do it at the whiteboard": Researcher-participant interaction and issues of generalisation. In *Proceedings of the Studying Professional Software Design (SPSD) Conference*. San Diego, CA, USA.

Rooksby, J., & Ikeya, N. (2012). Collaboration in Formative Design: Working Together at a Whiteboard. *IEEE Software*, *29*(1), 56 –60. doi:10.1109/MS.2011.123

Shaffer, D. W., & Resnick, M. (1999). "Thick" authenticity: New media and authentic learning. *J. Interact. Learn. Res.*, *10*(2), 195–215.

Smith, J. P., diSessa, A. A., & Roschelle, J. (1993). Misconceptions Reconceived: A Constructivist Analysis of Knowledge in Transition. *The Journal of the Learning Sciences*, *3*(2), 115–163.

Spacco, J., Hovemeyer, D., Pugh, W., Hollingsworth, J., Padua-Perez, N., & Emad, F. (2006). Experiences with Marmoset: Designing and Using an Advanced Submission and Testing System for Programming Courses. In *ITiCSE '06: Proceedings of the 11th annual conference on Innovation and technology in computer science education*. ACM Press.

Stevens, R., & Hall, R. (1998). Disciplined perception: Learning to see in technoscience. In *Talking Mathematics in School: Studies of Teaching and Learning* (pp. 107–150). Cambridge, U.K: Cambridge University Press.

Stieff, M. (2007). Mental rotation and diagrammatic reasoning in science. *Learning and Instruction*, *17*(2), 219–234. doi:10.1016/j.learninstruc.2007.01.012

Turkle, S., & Papert, S. (1990). Epistemological Pluralism: Styles and Voices within the Computer Culture. *Signs*, *16*(1), 128–157.

Acknowledgments

This project is supported in part by grants from the National Science Foundation (NSF): DRL 0733613 and EEC 0835880. The views expressed are those of the authors and not necessarily those of the NSF.