“I’m not a computer”: How identity informs value and expectancy during a programming activity

Kathryn Cunningham, Rahul Agrawal Bejarano, Mark Guzdial, Barbara Ericson
University of Michigan
kicunn@umich.edu, rahulab@umich.edu, mjguz@umich.edu, barbarer@umich.edu

Abstract: Code tracing—simulating the way the computer executes a program—is a common teaching and assessment practice in introductory programming courses. In a laboratory experiment where code tracing was encouraged, we found that some struggling novice programmers described code tracing as not only cognitively complex, but also in opposition to their self-beliefs. One participant described himself as not a computer, and therefore unfit to execute code like the computer does. Another described himself as not a programmer, and did not value an activity that was only for learning about how code works. We mapped these learners’ self-narratives onto the Eccles Expectancy-Value Model of Achievement Choice to understand how identity relates to the choice to not trace code. While both participants valued what they could create with code, neither valued code tracing. Alternative activities might allow students with these identities to build skills in a way that aligns with their self-beliefs.

Keywords: Code tracing, programming identity, expectancy-value theory, motivation

Introduction

Introductory programming activities often echo the epigram “*To understand a program you must become both the machine and the program*” (Perlis, 1982). *Code tracing* is a common learning and assessment activity where learners simulate program execution, describing the order in which lines of code run, how values are created and modified, and when the program starts and stops (Sorva, 2013). Tracing is an authentic practice for programming experts, who use it to debug programs (Vessey, 1985). In classroom settings, the ability to trace code unaided is correlated with the ability to write code and describe the purpose of code in natural language (Lopez, Whalley, Robbins, & Lister, 2008).

However, code tracing by hand is difficult, tedious, and novice programmers don’t always do it, even when it is a helpful strategy (Lister et al., 2004; Cunningham, Blanchard, Ericson & Guzdial, 2017). Learners describe the cognitive demands of tracing as one reason they try to avoid it by using alternative methods, like looking for familiar code patterns (Cunningham, Ke, Guzdial & Ericson, 2019). Hierarchies of programming skills describe code tracing as a primary ability that students naturally learn (Lister, 2011) or should be taught (Xie et al., 2018) before they learn to write code or explain the purpose of code. These hierarchies do not consider whether different students may have different values for different programming tasks.

In a laboratory setting, we asked undergraduate and graduate students with some prior programming experience to complete problems designed to encourage and isolate tracing behavior. To our surprise, some participants pushed back against instructions to trace code. Beyond avoiding tracing due to its difficulty, these participants described themselves as not the “type of people” who wanted to understand code tracing. We analyzed the attitudes, values, and beliefs these participants expressed, and mapped them onto the Eccles Expectancy-Value Model of Achievement Choice to explore: *How does identity relate to a novice programmer’s decision to not trace code?*

Identity and the Eccles Expectancy-Value Model of Achievement Choice

Why do learners choose to do some tasks, and not others? Drawing on prior work about decision-making and goal achievement, Jacquelynne Eccles and her colleagues proposed that two factors most directly influence the choice to select and complete a task: (1) a person’s *expectation for success* on the task and (2) a person’s *subjective value* for the task (Eccles, 1983). Expectation for success involves a person’s task-specific ability and self-concept, while subjective task value involves four factors: (a) attainment value, when the task aligns with the person’s self-image; (b) interest-enjoyment value, when the person expects to enjoy the task; (c) utility value, when completing the task helps the person achieve a goal; and (d) relative cost, where value is decreased due to the loss of other potential opportunities (Eccles, 2005). In Eccles’ model, a person’s *identity* directly influences *both* expectancy of success and subjective task value. Identity includes personal and social self-schemata, short and long-term goals, ideal self, and self-concept of general ability (Eccles, 2005).

Method

We conducted 30 minute interviews with 12 novice programmers (8 undergraduates and 4 PhD students at a large public research university) about code tracing tasks. Two interviewers (the first and second authors) were present at each interview. After each tracing task, participants answered semi-structured reflective questions about their problem-solving process. The final task of each interview involved a thinkaloud, where participants described their thoughts while they traced. Interviews were audio recorded and transcribed.

While the original intent of the interviews was to investigate problem-solving approaches, some participants provided unexpected feedback about their lack of motivation to trace code. We analyzed the transcripts to highlight the ways participants described their values and self-beliefs. We used Values Coding (LeCompte & Preissle, 1993; Saldaña, 2016), supplemented by In Vivo Coding (Charmaz, 2014; Saldaña, 2016) to preserve participants' tone and voice. Values Coding highlights the belief system of the participants by identifying the values (the importance of people, things or ideas), attitudes (the way participants think and feel about people, things, or ideas), and beliefs (rules and interpretations) in their talk. Using reflective summaries as an analysis tool, we mapped our themes onto the Eccles Expectancy-Value Model of Achievement Choice (Eccles, 2005).

Case studies

Two participants described most fully the reasons they choose to not trace when interacting with code. Both of these novice programmers expressed a strong negative opinion about our tracing tasks during their first round of problem-solving. As the interview continued, they provided well-developed descriptions about the reasons they found tracing tasks both difficult and not valuable. Mapping their values, attitudes, and beliefs onto elements of the Eccles Expectancy-Value Model of Achievement Choice creates a clearer picture of how the different parts of these learners' self-narratives interact and drive their opposition to code tracing tasks.

Charles: I'm not a computer

Charles is an undergraduate enrolled in his second semester of programming: a data-focused Python programming course offered by the university's iSchool. His prior experience included one prerequisite programming course at a community college.

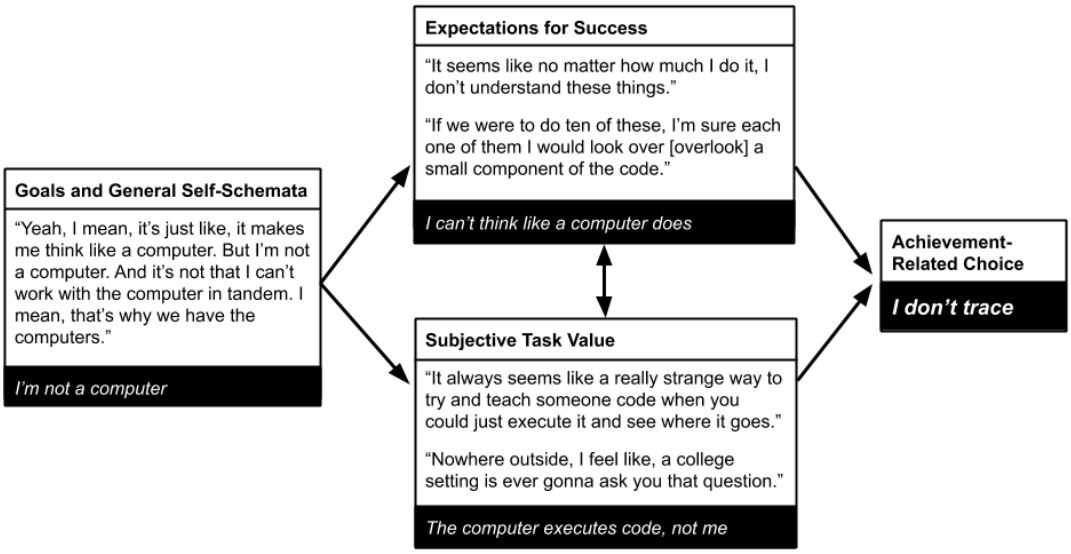

Figure 1. Charles' self-narrative mapped onto relevant aspects of the Eccles Expectancy-Value Model of Achievement Choice (Eccles, 2005).

Charles was fixed in his expectation that he would not succeed at code tracing. From his first opportunity to reflect on his problem-solving, Charles told us that understanding the role and importance of each piece of a program was *"the hardest thing for me with code in general*." Charles felt that would often "*jump to a conclusion"* about what code does, because he would "*[overlook] a small component of the code*." He did not believe be could improve in his code tracing skills, even with practice: "*It seems like no matter how much I do it, I don't understand these things*."

Charles doesn't value code tracing. Besides being cognitively costly, Charles described code tracing activities on past exams as *"demoralizing*" because he couldn't complete them accurately. Charles does value learning to program, but he described tracing as *"the least helpful thing for learning code."* Charles views code tracing tasks as inauthentic and a *"weird"* thing to be tested on, because "*nowhere outside, I feel like, a college setting is ever gonna ask you [to trace]*". Charles doesn't believe it's realistic to trace by hand, when he can simply use the computer to execute code.

At the end of the interview, Charles told us that he is "*not a computer*", and code tracing tasks make him "*think like a computer*". Tracing is simulating the computer's execution of code, and Charles is fixed in his belief that he cannot think like a computer thinks. Unlike a "hacker" who values thinking like a computer (Turkle, 1984), Charles doesn't believe he can read every detail of code and mentally execute it without error. Tracing is something the computer does when it executes code, but Charles is not a computer.

However, Charles doesn't count himself out of being someone who writes code ("*it's not that I can't work with the computer in tandem*"). Charles' goal is to complement the computer's strengths with his own.

Luke: I'm not a programmer

Luke is a PhD student in the university's iSchool who uses programming in his research. His prior programming experience included a formal programming course approximately three years in the past, as well as applied experience using code to create products used in research activities.

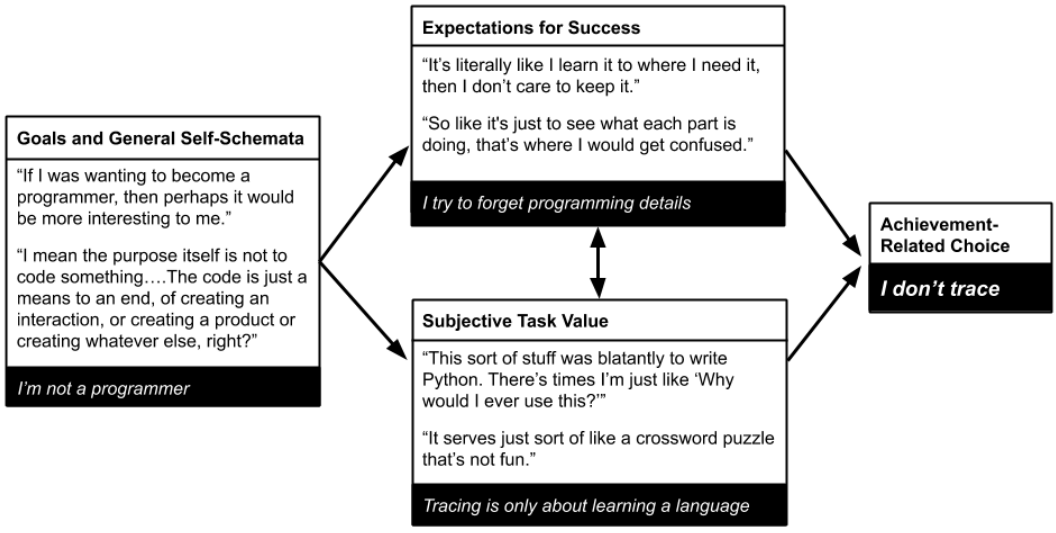

Figure 2. Luke's self-narrative mapped onto relevant aspects of the Eccles Expectancy-Value Model of Achievement Choice (Eccles, 2005).

Luke had low expectation of success in tracing exercises, due to a lack of understanding about "*what each part is doing*". Luke stated that this lack of skill was intentional, as he didn't "*care to keep"* programing knowledge that was "*no longer needed*" to achieve his immediate goals. However, he felt he would be able to recover tracing ability through study of "*some introductory thing on Python*".

Luke didn't see how completing code tracing activities could further his goals, or be enjoyable. For Luke, tracing problems are "*like a crossword puzzle that's not fun*", requiring high cognitive effort, but with no clear payoff. Context-less tracing problems about printing a square of asterisks and finding the largest item in a list didn't solve any problem Luke was interested in, and he asked "*Why would I ever use this?*" During one of the reflective portions of the interview, we asked Luke to try and describe the "purpose" of some code in one sentence. Luke deadpanned, "*teaching people Python*".

Like many end-user programmers, Luke achieves goals with programming, but does not identify as "*a programmer*" (Dorn & Guzdial, 2010). Code is only a "*means to an end*" for Luke, so he can create "*whatever it is I agree with, the design itself."* The details of code execution are not inherently valuable to Luke, and easily forgotten. Tracing is something that a programmer values, because it helps them understand how a programming language works, but Luke is not a programmer.

Conclusion

While previous research has focused on the ways that high cognitive load may influence the choice to trace code, our analysis identifies relationships between identity, expectations of success, and value for code tracing. We describe two identities that relate to choices about code tracing: *I'm not a computer* and *I'm not a programmer*. In our case studies, learners related these self-beliefs to a low expectation of success on code tracing, because they did not have the ability to notice code details or chose not to remember them. They also expressed a low value for code tracing, because it took a lot of effort, was not enjoyable, and did not appear relevant to their self-image and goals. Both Luke and Charles define themselves at a distance from people who are thinking deeply about how code works. However, from this distance, both participants see themselves as someone who uses programming. Charles is someone who can work *with* the computer, and Luke is someone who builds things *with* code.

Tracing code can certainly be difficult, intricate, and removed from the context of code writing. However, it is commonly positioned by computing education researchers as a "gateway" skill to programming expertise, frequently occupying the earliest rung in theorized pathways of programming learning (Lister 2011; Xie et al., 2019). For learners with the identities we describe, this code tracing "gate" is closed. Tracing is already so difficult that many computer science majors struggle with it (Lister et al., 2004). For learners who do not value understanding the mechanisms of code and intricacies of a language, there is little motivation to put the required effort into code tracing. If you see code as only a means to an end, why learn about code that doesn't have an application? If you want to work with the computer rather than be the computer, why simulate the machine?

Our learners are using code to achieve their goals, but they reject one of the most common activities in programming classrooms. To meet this type of programming learner where they are, we propose an exploration of programming learning activities that are function-oriented, contextualized, and authentic. While some programmers may not ever fully "become" the program or the machine, they can start by investigating what the program and the machine does for them.

References

Charmaz, K. (2006). Constructing Grounded Theory: A Practical Guide Through Qualitative Analysis. SAGE.

Cunningham, K., Blanchard, S., Ericson, B., & Guzdial, M. (2017). Using Tracing and Sketching to Solve Programming Problems: Replicating and Extending an Analysis of What Students Draw. *Proceedings of the 2017 ACM Conference on International Computing Education Research*, 164–172. ACM.

Cunningham, K., Ke, S., Guzdial, M., & Ericson, B. (2019). Novice Rationales for Sketching and Tracing, and How They Try to Avoid It. *In Innovation and Technology in Computer Science Education (ITiCSE '19)*.

Dorn, B., & Guzdial, M. (2010). Discovering computing: perspectives of web designers. In Proceedings of the Sixth International Workshop on Computing Education Research (pp. 23-30). ACM.

Eccles, J. (1983). Expectancies, values, and academic behaviors. In J.T. Spence (Ed.). Achievement and Achievement Motivations (pp. 75-121). San Francisco, CA: W. H. Freeman & Co.

Eccles, J. S. (2005). Subjective Task Value and the Eccles et al. Model of Achievement-Related Choices. *Handbook of Competence and Motivation* (1st ed.), 105-121. The Guilford Press.

LeCompte, M. D. and Preissle, J. (1993) *Ethnography and Qualitative Design in Educational Research* (2nd ed.). Academic Press.

Lister, R. (2011). Concrete and Other neo-Piagetian Forms of Reasoning in the Novice Programmer. Proceedings of the Thirteenth Australasian Computing Education Conference - Volume 114, 9–18.

Lister, R., Seppälä, O., Simon, B., Thomas, L., Adams, E., Fitzgerald, S., … Sanders, K. (2004). A multi-national study of reading and tracing skills in novice programmers. *ACM SIGCSE Bull.*, *36*, 119–150.

Lopez, M., Whalley, J., Robbins, P., & Lister, R. (2008). Relationships between reading, tracing and writing skills in introductory programming. In Proceedings of the Fourth International Workshop on Computing Education Research (pp. 101-112). ACM.

Perlis, A. J. (1982). Special feature: Epigrams on programming. ACM SIGPLAN Notices, 17(9), 7-13.

Saldaña, J. (2016). *The Coding Manual for Qualitative Researchers* (3rd ed.). Sage.

Sorva, J. (2013). Notional Machines and Introductory Programming Education. *Trans. Comput. Educ.*, *13*(2), 8:1–8:31.

Turkle, S. (2005). The Second Self: Computers and the Human Spirit. MIT Press.

Vessey, I. (1985). Expertise in debugging computer programs: A process analysis. International Journal of Man-Machine Studies, 23(5), 459-494.

Xie, B., Loksa, D., Nelson, G. L., Davidson, M. J., Dong, D., Kwik, H., … Ko, A. J. (2019). A theory of instruction for introductory programming skills. *Computer Science Education*, Vol. 29, pp. 205–253.