Work in Progress: Toward an Understanding of Strategies Used in Open-ended Programming Tasks

Darren K. Maczka, Virginia Tech

Darren Maczka is a Ph.D. candidate in Engineering Education at Virginia Tech. His background is in control systems engineering and information systems design and he received his B.S. in Computer Systems Engineering from The University of Massachusetts at Amherst. He has several years of experience teaching and developing curricula in the department of Electrical and Computer Engineering at Virginia Tech.

Dr. Jacob R. Grohs, Virginia Tech

Jacob Grohs is an Assistant Professor in Engineering Education at Virginia Tech with Affiliate Faculty status in Biomedical Engineering and Mechanics and the Learning Sciences and Technologies at Virginia Tech. He holds degrees in Engineering Mechanics (BS, MS) and in Educational Psychology (MAEd, PhD).

Work In Progress: Toward an understanding of strategies used in open-ended programming tasks

Introduction

Programming education is an increasingly common part of first year engineering curriculum.[1] However, the success of these efforts are fraught with challenges, both because teaching programming is hard in general,[2] and goals and motivations for learning programming are different for CS majors compared to non-majors[3,4] posing unique challenges for general first-year engineering programs that include prospective CS majors. It is this general first-year environment that is of interest to us and that inspired this study.

Our experiences with undergraduate general engineering education led us to identify computer programming tasks as a crucial component in one's identity as an engineer and in deciding how to participate in group projects. Anecdotaly we have observed that students who do not believe they have strong programming skills do not believe they can contribute to programming aspects of a project. As a result, the programming falls on the group member who identifies as a strong programmer. This is despite project and course setups that ostensibly promote a general level of programming competence for all students, and in particular, enough to successfully engage with the programming expected for the class project. Furthermore, computer programming is positioned in many first-year programs such that students associate programming skill with affinity towards computing majors (computer science of computer engineering). At universities with general first-year engineering programs, how students experience computer programming in the first year may have a significant impact in choosing whether or not to pursue a computing related major. This is particularly relevant in light of recent pushes to increase participation in computing fields coupled with observations that a feeling of "not belonging" contributes to some, in particular LGBTQ+[5] individuals and women,[6,7] not pursuing or leaving computing fields.

Pinning down just what skills are required to successfully program has been an ongoing debate since the dawn of programming itself. Abbate notes that the lack of understanding in just what skills were needed for programming manifested in the variety of job postings for programming positions. The skills listed on job postings tended to vary based on the industry, e.g. in a business setting programming was associated with data management and accounting skills while in a scientific setting it was associated with strong mathematical skills.[6] Though the skillset seems to have solidified if one looks at programming educational literature, it is also true that the variety of

positions requiring “programming” has similarly narrowed with the prevalence of commercial software packages designed for specific job sectors (e.g. secretarial work, accounting). Thus, in some ways “programming” has become more specialized than it perhaps once was.

The aim of this exploratory study is to discover skills and strategies that practicing programmers make use of that may not be explicitly taught or acknowledged in first-year programming courses. To achieve this goal we must utilize an appropriate theoretical perspective that accounts for the complex social and material interactions that a programmer engages with in practice.

Theoretical perspective

Because we are interested in studying the relationship between programmer and computer we need a theoretical perspective that incorporates both human and computer agencies. We chose the perspective of sociomateriality which has been used in organizational studies,[8] research of digital literacy in learning environments,[9] and increasingly in engineering education research.[10,11] Said simply, sociomateriality is the idea that humans and the material world can not be analyzed as disjoint entities. That is to say, if we want to understand a student’s cognitive process while programming we can not ignore the material world in which that student is situated and thinking. Interacting with tools, in particular the computer along with the specific software used, will shapes how we think and act.[12] While most studies that utilize the sociomaterial theoretical framework seek to describe system-level interactions, such as how the introduction of video-conferencing technology shifts power dynamics in a work group,[13] we take a slightly different focus. Because our ultimate goal is to improve the learning experience of individual students, we focus our data collection and inquiry to understand how individuals process and make meaning from their interactions in the sociomaterial world.

Methods and participants

This study is part of a more general exploratory study to investigate the utility of functional near-infrared spectroscopy (fNIRS) brain imaging technology for engineering education research. fNIRS is a non-invasive means of measuring cognitive activity in the surface areas of the brain. While we are still exploring techniques to analyzed the fNIRS data, we mention it here since the setup added specific constraints to the study environment.

After receiving IRB approval for human subjects research, we contacted the leaders of several computing-themed organizations at our university to distribute our invitation. These included technically-themed groups such as Unix/Linux Users groups, and cyber security themed groups, as well as affinity groups such as Women in Computing. We chose these organizations for the likelihood that members would have personal programming projects they could use for the study, and that by self-selecting into these groups, participants had at least some affinity for the field of computing. We asked leaders of these groups to distribute an invitation email to their members who would then contact us if they were interested in participating.

Once participants contacted a researcher we scheduled a time for them to come into a research lab on campus. When participants arrived, after reviewing the consent form and addressing any questions, we conducted a brief intake interview to learn about their experiences learning to program. After the intake interview, participants were fitted into an fNIRS cap and set up their computer for work. The cap is designed to fit snugly, and while effort is taken to make the fit as comfortable as possible, experience from a pilot study indicated the cap becomes increasingly irritating after 45-60 minutes of continuous use. For this reason, we planned to limit the time participants were wearing the cap to 45 minutes. Flexible wires connect the cap to the fNIRS machine which limits large movements, but does not restrict typical movements associated with programming, i.e. hand movement, leaning back while remaining to sit, etc.

As part of the setup process, a researcher started screen-capture recording hardware. Because we did not want to require that participants install special software we needed an external means to record screen capture data. For the first two participants we used a camcorder pointed at the participant's computer screen. The resulting video was only moderately useful as the camcorder did not consistently focus on the screen in the low light environment of the study location. For this reason, we ordered an HDMI screen recorder commonly used by gamers to record and share gaming sessions. We used this device for remaining participants, connecting it to their video-out port and asking them to mirror their screen to the recording device which was detected as an external monitor. We also used a stand alone audio recorder to continuously record from the intake interview through the outtake interview.

Once the necessary recording equipment was setup, participants started work on the project of their choice. Researchers generally did not interact with participants during their work session except to tell them when 45 minutes had passed so that they could reach a convenient stopping point. All participants signaled that they were done working within 5 minutes of the 45 minute notice.

After the work session, a research helped the participant remove the fNIRS cap and then conducted the outtake interview. The entire process from when the participant arrived at the study location until they left took about 1.25 to 1.75 hours.

Interview Protocol

The semi-structured interview protocol was designed to augment the data collected from screen capture and video recordings. The intake interview was designed to get a sense of participant's beliefs about programming, and beliefs about their skill at programming. The outtake interview was designed to learn possible points of interest in the programming session (e.g. times the participant got stuck) and to provide more context around the participant's access and experience working with computers growing up.

Intake

1. Please tell me your major and how you came to choose that major.

2. Describe what programming is to you.
    (a) What does it mean to "be a programmer"?
    (b) Do you consider yourself a programmer? Why or why not?
3. Describe how programming is integrated into the curriculum in your major.
    (a) What did you learn from the curricular activities vs. on your own?
4. Tell me about the project you will be working on today.
    (a) Why did you choose to start it?
    (b) How long have you been working on it?
    (c) How often do you work on it?
    (d) When do you think it will be complete?

Outtake

1. Think back over the programming session you just completed. At any time did you experience a "flow" state? At what point did you experience that?
    (a) If you didn't experience a flow state during this session, is it something you have experienced before? Can you describe the conditions that have led to it in the past?
2. Did you experience times of getting stuck or not knowing what to do next?
    (a) When did this occur, i.e. what made you stuck?
    (b) If you were able to resolve the situation, how did you?
3. Hypothetically, let's say you were starting a new programming project. Walk me through the first 15 minutes or so of that project.
4. I'm interested in knowing a little bit about your history with programming. Can you tell me how and when you first started programming?
5. Did you have your own computer growing up?
    (a) If not, did you have access to a shared (e.g. family) computer?
6. When you were first learning to program, what resources (including people) did you use?
7. When did you begin participating with your first computer-themed organization?
    (a) Did you feel welcome on the first day?
    (b) Why did you decide to stick with it (or not)?
    (c) Did you try participating in other organizations as well? Why or why not?

Preliminary analysis

Multi-modal data collection and analysis is not particularly new, nor are the specific modalities used here: interview audio, video recording, and screen capture. Our analysis method followed that used by Bhatt and Roock in their study of digital literacy events.[9] Our environment differed though, in an important regard. While their study was situated in the context of the classroom, and thus data were collected in that environment, we conducted our study in a research lab. This was a constraint of the study due to the data we wanted to collect, which included the use of an fNIRS machine located in a psychology lab on campus.

To prepare data for analysis we loaded the audio recording, facial video recording, and screen capture video recordings into ELAN, an open-source tool designed for time-aligned linguistic annotation.[14] Figure 1 shows a screen shot of ELAN during data analysis using a demonstration video of a member of the research team. Each of the three primary data streams: screen-capture video, front facing video, and audio, can be played back in sync using ELAN's controls. The lower part of the screen shows custom defined tiers, ELAN's term for an analytic layer of annotation. While not shown here, ELAN can also load and synchronize timeseries data such as eye tracking data, or in our case fNIRS data. We are still exploring suitable filtering and analysis of the fNIRS data to add useful information to the analysis process.

Preliminary analysis involved viewing several of the sessions within ELAN to get a sense of what to code for, and how to code for interactions between human participant and computer.[15] For an initial first-pass at coding we decided to code for compile error messages in the screen capture data and use those reference points to more closely explore interactions.

A notable pattern emerged even before coding for error messages: nearly every participant used a combination of the Google search engine and the StackOverflow website as part of their debugging process. The initial viewing indicates that most of the time, participants would enter the text of an error message into a Google search, and then select one of the top hits, usually linking to StakeOverflow. Participants tended to always go to Google first, despite almost always ending up at StackOverflow. Once on the StackOverflow site, some participants would refine their search if the first hit wasn't what they were looking for. A quick literature search indicated that the task of extracting useful information from StackOverflow results and applying it to one's own problem is a non-trivial task[16] suggesting that this process warrants further exploration.

Integrating neuroimaging data into analysis

Challenges remain with understanding how best to integrate the neuroimaging data collected with fNIRS into analysis. Our study design different significantly from other fNIRS and fMRI studies that use either block or event-related designs. In a block design participants are asked to engage in a short task, on the order of 10-15 seconds, such as studying a section of code on a screen, followed by a period of rest. Task-related data and rest-related data are averaged separately and across participants to determine if the task resulted in detectable activation in a particular region of the brain.[17] In an event-related design, tasks are also short, but are randomly presented to the participant. Data analysis techniques for both these designs assumes that the task is the same

across participants, and is relatively short compared to the time data were collected. In our design, however, neither of these assumptions hold. Participants are working for about 45 minutes on a single complex task, and while all participants are programming, there is tremendous variety across how each participant engages in the task.

We are currently evaluating exploratory data analysis methods that may be suitable for our study design. One such method is dynamic connectivity regression. This technique attempts to find time points at which connectivity across brain regions changes.[18] In particular, it may be suitable for situations when experimental tasks are not repeated, as in our setup.

Ongoing considerations

Defining the skillset necessary for "success" in a particular domain is not an objective task. Indeed, the concept of "success" itself is highly subjective and tends to shift with time and who and what ideas enjoy powerful positions in society. Often "skill" and "success" are defined by the tasks and achievements of the dominant group, programming being a prime example. Not only is this relevant to the study at hand (programming itself has been shaped by this social power, thus the participants we observe are products of that shaping), it is especially relevant when making decisions regarding how to synthesize these results into practice. Any changes to assessment must always be accompanied with reflection about how changes might affect different people, in particular those who have been historically disadvantaged. In short, we caution against rushing to

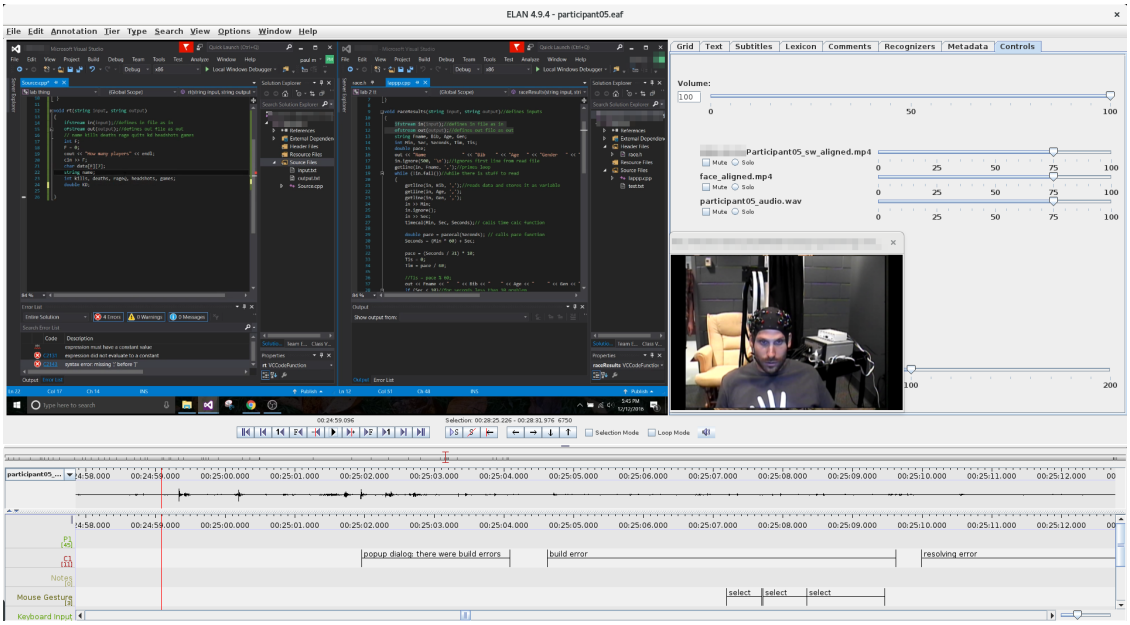

Figure 1: Screenshot of ELAN during data analysis. The large pane contains the screen capture video, the smaller window shows the front facing camera of a member of the research team for demonstration purposes. These two video streams, and the audio, are played in sync using the playback controls below the video panes. Below that we see the audio waveform and custom defined tiers, ELAN's term for a single analytic layer of annotations.

incorporate any initial findings, or those in the complete analysis, in your next programming assessment without further critique. This also serves as an important reminder to us as researchers that our findings are necessarily incomplete. By inviting participants who have been working on their own projects we are likely to get those that have more or less aligned their own identities and skillset with the dominate view of what programming is (with the important distinction that there may still be a disconnect between what we teach programming to be, and how it is generally practiced in non-academic environments). Thus, we are at risk of reinforcing the prevailing definition of what programming is and what skills are necessary to be successful at it by doing this research.

Open environments

Of particular interest in this study is moving beyond the confines of well-structured programming problems that we often present our students in class. Indeed, one participant from the study noted that the programming he did as part of his introductory engineering course was not "real" programming because "if they give you the solution to the problem, that's not problem solving, then I would argue that's not really programming either." This participant equated programming with problem solving, and believed that programming assignments in which the end goal were given, e.g. "write a program that does X", did not constitute problem solving, leading to the assessment that this was not "real" programming. It is interesting to note that this view is in stark contrast to the prevailing methods of analysis several decades ago when programming was often presented as simply the development of code that satisfied input/output data processing requirements that were handed to the programmer.[19] Thus, programming work has changed, and continues to change. While being aware of the limitations outlined above in assessing skill sets, we were interested in the aspects of programming practice that might not be captured in the conventional classroom environment. This motivated our study design to call for participants currently working on their own personal projects, rather than something assigned by a teacher or researcher. Further work is needed to learn how students transition from the highly structured programming environments of introductory courses to a more open environment. Ideally that work would inform introductory course design to help facilitate that transition.

Conclusion and future work

While our analysis is still ongoing, preliminary observation of the heavy use of external resources, such as internet search engines and crowd-sourced debugging help should prompt some reflection among programming instructors as to what skills we ought to teach and assess. For example, strategies for finding solutions to compile-time errors typically are not taught in introductory programming courses, but participants in this study all relied on skills to quickly determine which search result would help them solve their problem. In many cases, Google's algorithm put a relevant answer at the top or the results page, but not always. In some cases participants eliminated the first result without clicking on it, going immediately to the second. This suggests a practiced recognition of summaries as curated by Google.

A number of participants mentioned the complexity of the process they were attempting to implement with a program. However, few used external resources to help manage the complexity (e.g. diagrams, flowcharts) while most relied on "holding" the complexity in their head which some acknowledged required significant time to build up the complex mental representation before each session. This is of particular interest because some institutions teach the use of flowcharts as a tool to aid in visualizing logic flow in introductory classes, but these findings suggest that in practice flow charts may not be used to manage complex representations of processes. This warrants further investigation of just how expert programmers manage complex representations, and how this process might impact the cognitive load of novices. In particular, it suggests further inquiry as to why some experienced programmers choose not to use external aids to help maintain context in one's mind.

Our preliminary analysis suggests skills around error resolution such as interpreting error messages, identifying pieces of code with errors, and making use of online resources to find solutions to common errors are an important part of programming practice. Through further analysis we plan to finish categorizing the skills that are utilized by our participants, as well as make use of the fNIRS data we collected to incorporate information about cognitive load during different programming phases (adding a feature, debugging, testing a feature).

We caution against interpreting some of the observed tool use at face value. That is, even though nearly every participant utilized StackOverflow as tool in their process, this does not necessarily mean "teaching about StackOverflow" should be a specific objective in introductory programming courses. Specific tools change with time, and if contemporary tools are incorporated into the curriculum, they should be done so with attention to learning objectives and assessment techniques.[20] We encourage readers to think about the underlying skills that are involved in using a particular tool. Considering StackOverflow again, we find that using it requires a number of assumptions and beliefs. For example, the belief that a crowd-sourced forum can provide useful information to aid in debugging a specific error; the belief that one is capable of finding relevant information on a crowd-sourced repository and successfully interpreting it; the ability to identify relevant information in error messages and transform this into a search query that yields useful results. It is these skills and beliefs that we encourage researchers to explore further, and that we plan to explore through the completion of this study.

References

[1] Kenneth Reid and David Reeping. "A Classification Scheme for "Introduction to Engineering" Courses: Defining First-Year Courses Based on Descriptions, Outcomes, and Assessment". In: *American Society for Engineering Education Annual Conference & Exposition. Indianapolis, IN (1-11). Washington DC: American Society for Engineering Education*. 2014.

[2] Theodora Koulouri, Stanislao Lauria, and Robert D. Macredie. "Teaching Introductory Programming: A Quantitative Evaluation of Different Approaches". In: *ACM Transactions on Computing Education (TOCE)* 14.4 (2014), p. 26.

[3] Andrea Forte and Mark Guzdial. "Motivation and Nonmajors in Computer Science: Identifying Discrete Audiences for Introductory Courses". In: *Education, IEEE Transactions on* 48.2 (2005), pp. 248–253.

[4] Mark Guzdial and Andrea Forte. "Design Process for a Non-Majors Computing Course". In: *ACM SIGCSE Bulletin*. Vol. 37. ACM, 2005, pp. 361–365.

[5] J.G. Stout and H.M. Wright. “Lesbian, Gay, Bisexual, Transgender, and Queer Students’ Sense of Belonging in Computing”. In: *Research in Equity and Sustained Participation in Engineering, Computing, and Technology (RESPECT), 2015*. Research in Equity and Sustained Participation in Engineering, Computing, and Technology (RESPECT), 2015. Aug. 2015, pp. 1–5. DOI: 10.1109/RESPECT.2015.7296501.

[6] Janet Abbate. *Recoding Gender: Women’s Changing Participation in Computing*. MIT Press, 2012.

[7] Anna Vitores and Adriana Gil-Juárez. “The Trouble with ‘Women in Computing’: A Critical Examination of the Deployment of Research on the Gender Gap in Computer Science”. In: *Journal of Gender Studies* 25.6 (Nov. 1, 2016), pp. 666–680. DOI: 10.1080/09589236.2015.1087309.

[8] Wanda J. Orlikowski and Susan V. Scott. “Sociomateriality: Challenging the Separation of Technology, Work and Organization. The Academy of Management Annals 2 (1), 433–474”. In: *The Academy of Management Annals* 2.1 (2008), pp. 433–474. DOI: 10.1080/19416520802211644.

[9] Ibrar Bhatt and Roberto de Roock. “Capturing the Sociomateriality of Digital Literacy Events”. In: *Research in Learning Technology* 21 (Jan. 2014). DOI: 10.3402/rlt.v21.21281.

[10] Aditya Johri. “The Socio-Materiality of Learning Practices and Implications for the Field of Learning Technology”. In: *Research in Learning Technology* 19.3 (Sept. 2011), pp. 207–217. DOI: 10.1080/21567069.2011.624169.

[11] Aditya Johri. “Sociomaterial Bricolage: The Creation of Location-Spanning Work Practices by Global Software Developers”. In: *Information and Software Technology*. Studying work practices in Global Software Engineering 53.9 (Sept. 2011), pp. 955–968. DOI: 10.1016/j.infsof.2011.01.014.

[12] Sylvia Scribner and Ethel Tobach. *Mind and Social Practice: Selected Writings of Sylvia Scribner*. Google-Books-ID: ppTiqXHfhAYC. Cambridge University Press, Jan. 13, 1997. 408 pp.

[13] Suzie Wong Scollon. *Nexus Analysis: Discourse and the Emerging Internet*. Routledge, 2004.

[14] Peter Wittenburg et al. “Elan: A Professional Framework for Multimodality Research”. In: *Proceedings of LREC*. Vol. 2006. 2006, 5th.

[15] Hennie Brugman, Albert Russel, and Xd Nijmegen. “Annotating Multi-Media/Multi-Modal Resources with ELAN.” In: *LREC*. 2004.

[16] Mohammad Masudur Rahman and Chanchal K. Roy. “Recommending Relevant Sections from a Webpage about Programming Errors and Exceptions”. In: *Proceedings of the 25th Annual International Conference on Computer Science and Software Engineering*. IBM Corp., 2015, pp. 181–190.

[17] Scott A. Huettel, Allen W. Song, and Gregory McCarthy. *Functional Magnetic Resonance Imaging*. Vol. 1. Sinauer Associates Sunderland, 2004.

[18] Ivor Cribben et al. “Dynamic Connectivity Regression: Determining State-Related Changes in Brain Connectivity”. In: *NeuroImage* 61.4 (July 16, 2012), pp. 907–920. DOI: 10.1016/j.neuroimage.2012.03.070.

[19] Ruven Brooks. “Towards a Theory of the Cognitive Processes in Computer Programming”. In: *International Journal of Man-Machine Studies* 9.6 (Nov. 1977), pp. 737–751. DOI: 10.1016/S0020-7373(77)80039-4.

[20] Darren K Maczka and Jacob R Grohs. “Leveraging Historical Ties Between Cognitive Science and Computer Science to Guide Programming Education”. In: *2016 ASEE Annual Conference and Exposition*. ASEE Annual Conference and Exposition. New Orleans, LA, June 2016.