International Society of the Learning Sciences

A Duoethnographic Study of a Mixed-Ability Team in a Collaborative Group Programming Project

Mina Huh, KAIST, minarainbow@kaist.ac.kr
JooYoung Seo, University of Illinois at Urbana-Champaign, jseo1005@illinois.edu

Abstract: As diversity efforts in computer science education strive to make programming more accessible, we take a step forward by exploring the collaborative context. We present a duoethnography of a mixed-ability team collaborating on a group programming project. We (one sighted and one blind individual) immersed ourselves in a collaborative programming project using a social coding platform (i.e., GitHub) and triangulated our self-reflexivity through cross-checking reviews and interviews. By analyzing the interaction logs on GitHub, code writing in integrated development environments, and interview notes, we show the distinct approaches adopted by the sighted programmer and the programmer with visual impairments, as well as the social implications of visual impairment. This research offers reflections on the accommodations and technologies provided to support mixed-ability teams collaborating on a group programming project.

Introduction

Since Wing's remark on "computational thinking" in 2006, computer science (CS) education has been increasingly emphasized across formal and informal settings to engage learners in creative problem solving (Yadav et al., 2016) and provide scaffolding opportunities for career development in STEM disciplines (Webb et al., 2017). In the coding education context, project-based learning (Krajcik & Shin, 2014) has been widely adopted as a CSCL research legacy within a small-group unit, where two or more students work collaboratively on a shared programming challenge to promote group cognition (Stahl, 2006). Group projects offer opportunities for learners to hone collaborative skills in planning, time management, and refining and explaining code (Burke, 2011). However, this type of learning paradigm can pose challenges to learners with disabilities when not carefully designed. For example, Seo et al. (2017) discussed how inaccessible technologies impact mixed-ability group performance and dynamics between blind and sighted teammates in on-/offline hybrid CSCL settings. On the other hand, universally designed group work allows learners with diverse abilities to strongly engage in inclusive collaboration ethics (Pires et al., 2020).

The purpose of this study was to explore the practices and challenges of a mixed-ability team in a small-group collaborative programming project through a duoethnographic approach. As a mixed-ability team, we (one author is blind; the other is sighted) immersed ourselves in a collaborative programming project using a popular social coding platform (i.e., GitHub) and triangulated our self-reflexivity through cross-checking interviews. We analyzed the interaction logs on GitHub, recordings of code writing in integrated development environments (IDEs), and reflection notes. This research highlights the difficulties that programmers with visual impairments face and the positive ways in which technology is alleviating certain barriers.

Methods

Duoethnography

We used duoethnography, which is a qualitative research method in which two or more researchers provide different meanings to a common experience. Duo-/autoethnography builds upon ethnography and autobiography by using self-reflection to connect personal experiences to wider understandings (Ellis et al., 2011). In the accessibility domain, researchers have used these methods to generate personal insights into people with disabilities. To understand the experiences of a mixed-ability team in virtual settings, Mack et al. (2021) conducted a group autoethnography of a virtual internship. While similar to autoethnography, duoethnography facilitates dialectical interactions between researchers. In this paper, we adopted duoethnography to uncover insights into the experiences of a mixed-ability group through the comparison and analysis of differences.

Biography

The first author, Huh is an individual without a visual impairment who has 5 years of programming experience. She has worked on multiple programming assignments as part of a team but had never been in a mixed-ability team before this study. She is familiar with assistive technologies, but does not regularly use them. The other

author, Seo is an individual with a visual impairment (self-identified as blind), who has more than 10 years of programming experience. As a self-taught programmer, he started coding when he was 12 years old. Going through countless accessibility barriers, he had gradually come up with non-visual coding strategies using screen readers and a refreshable braille display. While he had a few collaborative coding experiences with sighted programmers through open-source contributions on GitHub and his engineering internship, this study allowed him an immersive experience with a mixed-ability group programming project similar to a CS education setting.

Data collection and analysis

The authors participated in a 6-day think-aloud study while collaborating on a programming task. The study was conducted remotely, and the authors worked asynchronously. In total, there were five sessions (see Table 1), each denoting a non-interrupted turn by a single author. All sessions were self-instructed. Due to the cognitive load imposed by simultaneously writing codes and articulating thoughts, each session was completed with retrospective note-taking. For the programming task, we selected an assignment from an introductory programming course at a local university in South Korea. The selection criteria were the expected duration to finish the task, the authors' familiarity with the programming language used, and the task complexity required to yield sufficient back-and-forth communications. The recordings of think-aloud sessions and interviews, retrospective notes, and interaction logs in IDEs and GitHub were analyzed. We used open coding to articulate the findings presented below.

Table 1: *Details of each study session*

	Author	Commit/issue message	Lines changed
Session 1	Huh	Create README.md	+69
	Huh	Update README.md	+1 -1
	Huh	Task 1 done	+21
Session 2	Seo	Prettify formatting	+1 -2
	Seo	Cosmetic change: variable names and comments	+16 -11
	Seo	Add a new function: drawing_integers()	+14
	Seo	Add a new function: average_integers()	+15
Session 3	Huh	Added condition check for functions in draw_integer file	+27 -2
	Huh	Implemented count_integers	+20 -3
Session 4	Seo	[Issue]: Return an empty list in drawing_integers function	No change
Session 5	Huh	Replace if-else statements to try-catch statements for input check	+23 -20

Findings

Information access

The findings of the study revealed the distinct practices adopted by the two authors in accessing references. For instance, when accessing the project description, Huh referred to the slide-format PDF file, whereas Seo used the Markdown file that she added. She noted the benefits of the PDF file since it has more intuitive visuals, such as vivid colors, fonts, and diagrams. Also, the provided PDF file was created using slides, and the description was split into smaller pages that contained 10–12 lines of description, on average. Huh found it helpful since each slide unit would better fit the size of a computer screen, while the segmentation of information helped her to better concentrate on the content. This echoes the findings of Mayer et al. (2003), which noted that knowledge segmentation reduces the cognitive load in multimedia learning. Conversely, Seo accessed descriptions in the Markdown file created as a README file by Huh. Seo mentioned, "Based on my past experiences, 90% of PDF files are not accessible." In most cases, the PDF manuals that I get on the Internet don't conform to accessibility standards, being either untagged or non-navigable scanned files. Some modern screen readers, such as JAWS and NVDA, provide OCR (optical character recognition) to translate image PDFs into navigable text objects. But, that's not 100% accurate. Plus, the OCR often misses some semantic document structures and formatting, which is critical."

When writing codes, both authors occasionally searched online to understand a programming concept, review Python grammar, or look up built-in functions. Different strategies were also observed in accessing online search results. Huh noted she preferred checking the image results of the search query. She stated, "Though it may be a personal preference, I prefer to check the image results first because I can check dozens of

results quickly by skimming what their image contents show. It's like a preview thumbnail of each link to a website and is more efficient than having to visit each website from the top until I find the one I like." On the contrary, Seo preferred to visit technical forums (e.g., Stack Overflow and W3Schools references) in his Google search results. He said, "These are reliable sites where I can quickly find my answers in a fully-accessible text form. In Stack Overflow, all the comments are marked with semantic headings so that I can skip over to the next replies by pressing the single-letter navigation key "H" using a screen reader. The W3Schools website is designed with accessibility principles in mind, so this is one of my go-to references."

Code review

Both authors started reviewing codes by checking the commit messages left by their collaborator. When the message was clear enough to guess the changes, both of them skipped comparing the new code to the previous version. However, when the commit message was not self-explanatory, or when the codes were added to core parts of the project, they reviewed the changes using Git's tracking change features. Seo used a command line interface (CLI) by entering keyboard commands such as "git diff HEAD~" and "git log --oneline" to differentiate the insertion and deletion of codes denoted by prefix +/- signs. However, Huh preferred to use GitHub's graphical user interface (GUI) to compare the two versions of codes. One reason for choosing a GUI over a CLI was that the buttons in GUIs were easier to learn in comparison to remembering commands. Huh noted, "I am not good at memorizing commands and option flags. When I visit the repository website, I can just click the menus!" She also commented on the convenience of having a split view in addition to a unified view, where she could compare two codes in parallel. Huh mentioned, "Even if the insertion and deletion are differentiated visually in the unified view, it's hard to picture what the resulting code would look like when I view the actions line by line."

Tools and technology

Both authors used the same type of IDE—Visual Studio Code (VSCode). Seo used the JAWS screen reader to access the IDE, the internet, and the terminal. He mentioned that different assistive technologies provide different accessibility and user experiences. For instance, NVDA has more features than JAWS for customizing various sound schemes in a programming context, such as playing different sound effects according to the level of indentation and type of punctuation. Also, a braille display can be effective when using programming languages with indentations in code blocks.

The preferred interface for GitHub interactions (e.g., adding new commits) also differed. Huh used the terminal integrated into IDE to view everything in the same window. In contrast, Seo opened an external terminal (i.e., CMD.exe) in a separate window to keep the two tasks (writing programming codes in editor and executing commands in console) cognitively distinct. Seo noted, "Sighted people can easily manage the editor and console panes in the same window. But, I, as a blind programmer, prefer a more discernible instance split. It's more efficient and less error prone for me to switch the system focus between separate windows by pressing Alt+Tab key than to move my screen reader focus between different sub-panes within the same window."

As strategies for learning new tools, both authors mentioned that they refer to the official documents. In addition to text-based references, Huh mentioned the benefits of video-based learning. "These days, any tutorial is on YouTube. It's easier to learn with videos since they visually demonstrate how to interact with the software while narrating the explanations." For learning how to control the coding environments (e.g., git/GitHub, VSCode), Seo compared his experiences of navigating GUI features with screen readers and CLI workarounds. He noted, "I like to use CLI instead of GUI because typing commands and getting results are much quicker than navigating GUI elements. Of course, the learning curve is steep for CLI since I have to learn and memorize each command. Usually, I make my own command-line cheatsheets for each software I use." Additionally, Seo mentioned the difficulty in troubleshooting the accessibility-related issues of tools. To find solutions, he said that he often reaches out to a tool expert in the blind programmers' community and contact the accessibility support team of the company because technical forums such as Stack Overflow often lack relevant information.

Social implications of visual impairment

Both authors adopted certain practices for the convenience of the other collaborator. For instance, when creating the repository, Huh wrote down all the task descriptions in the PDF file in both Markdown format and as comments as part of the Python files. She stated, "Even though Seo didn't ask for it, I converted the PDF descriptions into Markdowns and comments. I didn't know which format is the most accessible. Thus, I provided several options so that he can choose." Seo used the auto-formatting feature in VSCode before pushing

his Python code into the repository to improve code readability for his sighted collaborator. Through cross-feedback, the quality of codes in terms of performance and documentation improved over time. While the authors did not use any other communication medium, they used contextual commit messages to share updates in the code. For implicit feedback, the authors corrected their collaborator's code to guide the coding style. For explicit feedback, one author (Seo) created an issue using GitHub that Huh later reflected on and resolved. When asked the reason for skipping discussions to reach an agreement, both authors mentioned the capability to predict the other collaborator's needs. Huh stated, "If both of us had no prior knowledge in accessibility, we would have had to communicate more often to get the sense of the other's experience and needs."

Discussions, limitations, and future work

From this study, we uncovered numerous barriers experienced by programmers with visual impairments. We believe that efforts from educators, students, and researchers can help remove these barriers. First, better quality accommodations should be provided in CS education with each component (e.g., lectures, learning materials, and assignments). As noted by Baker et al. (2019), inaccessible assignments increase the workload and interfere with learning. Also, more technologies should be designed to support collaboration in programming. While there is a rich thread of technical research that supports blind programmers in writing codes, the use of collaboration tools such as GitHub has been underexplored in this regard. The current method of tracking code changes heavily relies on visual elements such as colors, fonts, and +/- signs. We believe that more intuitive audio representations of code insertions and deletions will improve code-reviewing experiences for blind programmers. We also have limitations for the study generalizability. Although we used duoethnography as a rich and rigorous first-person method, this method relies heavily on the authors' personal experiences and unique viewpoints. In future work, we will extend the number of interviewees and also investigate different experiences based on various types of visual impairments and experiences with programming.

Conclusion

This paper outlined the practices and challenges of mixed-ability teams collaborating on a group programming project using a duoethnographic study. We explored the distinct approaches adopted by a sighted programmer and a programmer with visual impairments, the tools and technologies used, and the implications of visual impairment in terms of learning and the social context. Overall, our findings and discussions shed light on future research in developing assistive technologies to help people with visual impairments in collaborative programming.

References

Ellis, C., Adams, T. E., & Bochner, A. P. (2011). Autoethnography: An overview. *Historical Social Research / Historische Sozialforschung*, 36(4 (138)), 273–290.

Mack, K., Das, M., Jain, D., Bragg, D., Tang, J., Begel, A., Beneteau, E., Davis, J. U., Glasser, A., Park, J. S., & Potluri, V. (2021). Mixed Abilities and Varied Experiences: A group autoethnography of a virtual summer internship. *The 23rd International ACM SIGACCESS Conference on Computers and Accessibility*, 1–13.

Mayer, R. E., & Moreno, R. (2003). Nine ways to reduce cognitive load in multimedia learning. *Educational Psychologist*, 38(1), 43–52.

Pires, A. C., Rocha, F., de Barros Neto, A. J., Simão, H., Nicolau, H., & Guerreiro, T. (2020). Exploring accessible programming with educators and visually impaired children. *Proceedings of the Interaction Design and Children Conference*, 148–160.

Seo, J., AlQahtani, M., Ouyang, X., & Borge, M. (2017). Embracing Learners With Visual Impairments in CSCL. *12th international conference on computer supported collaborative learning* (Vol. 2, pp. 573–576).

Stahl, G. (2006). Group cognition: Computer support for building collaborative knowledge. Cambridge, MA: MIT Press.

Webb, M., Davis, N., Bell, T., Katz, Y. J., Reynolds, N., Chambers, D. P., & Sysło, M. M. (2017). Computer science in K-12 school curricula of the 2lst century: Why, what and when? *Education and Information Technologies*, 22(2), 445–468.

Wing, J. M. (2006). Computational thinking. *Communications of the ACM*, 49(3), 33–35.

Yadav, A., Hong, H., & Stephenson, C. (2016). Computational thinking for all: Pedagogical approaches to embedding 21st century problem solving in k-12 classrooms. *TechTrends*, 60(6), 565–568.