International Society of the Learning Sciences

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

Author names omitted due to the blind review

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

Introduction

Since Wing's remark on "Computational Thinking" in 2006, computer science (CS) education has been more emphasized across formal and informal settings to engage learners in creative problem solving (Yadav et al., 2016) and to provide scaffolding opportunities for career development in STEM disciplines (Webb et al., 2017). In coding education context, project-based learning (PBL; Krajcik & Shin, 2014) has been widely adopted as a CSCL research legacy within a small-group unit, where two or more students collaboratively work 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 as well as refining and explaining the code (Burke, 2011); however, such type of learning paradigm can pose challenges to learners with disabilities when not carefully designed. For example, author 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 is to explore the practices and challenges of a mixed-ability team in a small-group collaborative programming project through a duo-ethnographic approach. We, as a mixed-ability team (one author is blind; the other is sighted), immersed ourselves into 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 (IDE), and reflection notes. This research highlights the difficulties that programmers with visual impairments face, and the positive ways how technology is alleviating some of the barriers. It also raises key considerations for accommodations and designing technologies for programmers with visual impairments.

Methods

Duoethnography

We used duoethnography, a qualitative research method in which two or more researchers provide different meanings to a common experience. Duo-/autoethnography build upon ethnography and autobiography by using self-reflection to connect personal experience to wider understandings (Ellis et al., 2011). In accessibility domain, researchers have used these methods to generate personal insights of people with disabilities. To understand 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 mixed-ability group through comparison and analysis of difference.

Biography

One author, M is an individual without a visual impairment who has 5 years of programming experience. She has worked on multiple programming assignments as a team, but has never been in a mixed-ability team before this study. As an accessibility researcher, she is familiar with types of assistive technologies, but has not used them. The other author, J 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 of a mixed-ability group programming project similar to a CS education setting.

Data collection and analysis

The authors participated in a six-days think-aloud studies while collaborating on a programming task. The study was conducted remotely and authors worked asynchronously. In total, there were 5 sessions (see Table 1), each denoting a non-interrupted turn by a single author. All sessions were self-instructed, and because of the cognitive load imposed by writing codes and articulating thoughts at the same time, each session was completed with a retrospective note taking. For the programming task, we selected assignment of an introductory programming course from 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 to yield enough back-and-forth communications. The recordings of think-aloud sessions and interviews, retrospective notes, as well as interaction logs in IDE and GitHub were anlayzed. We used open coding to articulate below findings.

Table 1
Information details of each study session

	Author	Commit/issue message	Lines changed
Session 1	M	Create README.md	+69
	M	Update README.md	+1 -1
	M	task1 done	+21
Session 2	J	Prettify formatting	+1 -2
	J	Cosmetic change: variable names and comments	+16 -11
	J	Add a new function: drawing_integers()	+14
	J	Add a new function: average_integers()	+15
Session 3	M	added condition check for functions in draw_integer file	+27 -2
	M	implemented count_integers	+20 -3
Session 4	J	[Issue]: Return an empty list in drawing_integers function	No change
Session 5	M	replaced if-else statements to try-catch statements for input check	+23 -20

Findings

Information access

Findings from the study revealed the distinct practices adopted by the two authors in accessing references. For instance, when accessing the project description, M referred to the slide-format PDF file originally provided by the course staff whereas J used the Markdown file added by M. M noted the benefits of PDF file as it having more intuitive visuals such as vivid colors, fonts and diagrams. Also, the provided PDF was created using slides, and the description was splitted into smaller pages that in average contained 10-12 lines of description. M found it helpful as each slide unit fitted the size of a computer screen better, and the segmentation of information helped her to better concentrate on the content. This echoes the findings of Mayer et al. (2003) that knowledge segmentation reduces the cognitive load in multimedia learning. Conversely, J accessed descriptions in the Markdown file created as README file by M. J mentioned "Based on my past experiences, 90% of PDF files are not accessible. In most cases, 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 pdf 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, to review Python grammars, or to lookup built-in functions. Different strategies were also observed in accessing online search results. M noted she preferred checking 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 website and

definitely more efficient than having to visit each website from top unitil I find the one I like." On the contrary, J 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 single-letter navigation key "H" using screen reader. 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 the collaborator. When the message was clear and provided sufficient information 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 collaborator pushed codes on core parts of the project, they reviewed the incoming changes using Git's tracking change features. J used Command Line Interface (CLI) for code comparison. He enters keyboard commands, such as "git diff HEAD~" and "git log --oneline" to differentiate the insertion and deletion in codes denoted by prefix +/- signs. M, however, preferred to use GitHub's Graphical User Interface (GUI) to compare two versions of codes. One reason for choosing GUI over CLI was that buttons in GUIs were easier to learn compared to remembering commands. M noted "I am not good at memorizing commands and option flags. When I visit therepository 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. M 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). J used JAWS screen reader to access IDE, Web, and the terminal. He mentioned that different assistive technologies provide different accessibility and user experiences. For instance, NVDA has more features than JAWS to customize various sound schemes in a programming context, such as playing different sound effects according to level of indentation and type of punctuation. Also, braille display can be effective when using programming languages with indentations in code blocks. In this study, J used JAWS for its compatibility with Zoom which was used for screen-recording.

Preferred interface for GitHub interactions (e.g., pushing and pulling new commits) also differed. M used the terminal integrated in IDE to view everything in the same window. In contrast, J opened a terminal popup in a separate window to keep the two tasks (writing codes and wring commands) cognitively distinct. J noted "When sighted you can differentiate different sectionsin the same window, but as a blind programmer I have to keep switching the focus of my screen reader when switching the type of tasks. It's better to switch windows, so that there is no error caused from continuous tabs in the attempt of switching focus in 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, M mentioned the benefits of video-based learning. "These days, any tutorial is on YouTube. It's easier to learn with videos as they visually demonstrate how to interact with the software while narrating the explanations." For learning how to control the software, J compared experiences of navigating features with screen readers and typing commands in the terminal. He noted "Learning curve can be steep for commands-based controls as I have to learn and memorize each command. Usually, I keep a list of frequently used commands for each software I use in a notepad. Navigating elements (buttons) can be easier when using the tool for the first time, but it'd be cumbersome if I have to linearly navigate every time." In addition, J mentioned the difficulty of troubleshooting accessibility related issues of tools. To find solutions, he reached out to the tool expert in the blind programmers' community, or contacted the accessibility support team of the company because technical forums such as Stack Overflow often lacked the 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, M wrote down all the task description in the PDF file into both Markdown format and comments as part of python files. She stated "Even though J 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 provided several options so that he can choose." J used auto-formatting feature in VSCode before pushing his Python code into the repository to improve the code readability for the sighted collaborator.

Through cross-feedback, the quality of codes in terms of performance and documentation improved over time. Authors did not use any other communication medium but used contextual commit messages to share the updates in the code. For implicit feedback, authors corrected the code of the other to guide the coding style. For explicit feedback, one author, J created an issue using GitHub which M later reflected 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. M 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 the study, we uncovered numerous barriers that programmers with visual impairments undergo. We believe that efforts from educators, students, and researchers can help removing these barriers. First, better quality of accommodations should be provided in CS education with each component; lectures, learning materials and assignments. As Baker et al. (2019) noted, 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, collaboration tools such as GitHub has been underexplored.Current way of tracking code changes highly relies on visual elements such as colors, font s, and +/- signs. We believe that more intuitive audio representations of code insertions and deletions will improve the code-reviewing experiences for blind programmers.

In this research, we used duoethnography as rich and rigorous first-person method. Yet, we acknowledge that duoethnographic approach can be limited as it assumes the authors' willingness to disclose oneself honestly. As the future work, we will extend the number of interviewees, and also look at different experiences from participants' varied type of visual impairments and experiences with programming.

Conclusion

This paper shows the practices and challenges of mixed-ability teams collaborating in a group programming project using duoethnographic study. We explored distinct approaches adopted by the sighted programmer and the programmer with visual impairments, tools and technologies used, and implications of visual impairment in learning and 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

Author et al. (2017). Blind for review.

Baker, C. M., Bennett, C. L., & Ladner, R. E. (2019, February). Educational experiences of blind programmers. In Proceedings of the 50th ACM Technical Symposium on Computer Science Education (pp. 759-765).

Burke, A. (2011). Group work: How to use groups effectively. Journal of Effective Teaching, 11(2), 87-95.

Ellis, C., Adams, T. E., & Bochner, A. P. (2011). Autoethnography: an overview. Historical social research/Historische sozialforschung, 273-290.Brown, A. (1992). Design experiments: Theoretical and methodological challenges in creating complex interventions in classroom settings. *The Journal of Learning Sciences*, *2*(2), 141–178.

Mack, K., Das, M., Jain, D., Bragg, D., Tang, J., Begel, A., ... & Potluri, V. (2021, October). Mixed Abilities and Varied Experiences: a group autoethnography of a virtual summer internship. In The 23rd International ACM SIGACCESS Conference on Computers and Accessibility (pp. 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, June). Exploring accessible programming with educators and visually impaired children. In Proceedings of the Interaction Design and Children Conference(pp. 148-160).

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.