The Impact of Group Discussion and Formation on Student Performance: An Experience Report in a Large CS1 Course

Tong Wu
Virginia Tech
Blacksburg, VA, USA
tongw@vt.edu

Xiaohang Tang
Virginia Tech
Blacksburg, VA, USA
xiaohangtang@vt.edu

Sam Wong
University of Washington
Seattle, WA, USA
samw627@uw.edu

Xi Chen
University of Virginia
Charlottesville, VA, USA
xic@vt.edu

Clifford A. Shaffer
Virginia Tech
Blacksburg, VA, USA
shaffer@cs.vt.edu

Yan Chen
Virginia Tech
Blacksburg, VA, USA
ych@vt.edu

ABSTRACT

Programming instructors often conduct collaborative learning activities, such as Peer Instruction (PI), to enhance student motivation, engagement, and learning gains. However, the impact of group discussion and formation mechanisms on student performance remains unclear. To investigate this, we conducted an 11-session experiment in a large, in-person CS1 course. We employed both random and expertise-balanced grouping methods to examine the efficacy of different group mechanisms and the impact of expert students' presence on collaborative learning. Our observations revealed complex dynamics within the collaborative learning environment. Among 255 groups, 146 actively engaged in discussions, with 96 of these groups demonstrating improvement for poor-performing students. Interestingly, our analysis revealed that different grouping methods (expertise-balanced or random) did not significantly influence discussion engagement or poor-performing students' improvement. In our deeper qualitative analysis, we found that struggling students often derived benefits from interactions with expert peers, but this positive effect was not consistent across all groups. We identified challenges that expert students face in peer instruction interactions, highlighting the complexity of leveraging expertise within group discussions.

CCS CONCEPTS

• **Social and professional topics** → **Computing education**.

KEYWORDS

Collaborative learning; Peer instruction; Group mechanisms

ACM Reference Format:
Tong Wu, Xiaohang Tang, Sam Wong, Xi Chen, Clifford A. Shaffer, and Yan Chen. 2025. The Impact of Group Discussion and Formation on Student Performance: An Experience Report in a Large CS1 Course. In *Proceedings of the 56th ACM Technical Symposium on Computer Science Education V. 1 (SIGCSE TS 2025), February 26-March 1, 2025, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3641554.3701973

SIGCSE TS 2025, February 26-March 1, 2025, Pittsburgh, PA, USA

ACM ISBN 979-8-4007-0531-1/25/02.
https://doi.org/10.1145/3641554.3701973

1 INTRODUCTION

Collaborative learning has become an integral component of computer science education, particularly in introductory programming courses (CS1) [11]. This approach actively engages students in working together, learning new concepts, solving problems, and providing peer feedback. Programming instructors often employ various collaborative learning activities, such as Peer Instruction [9], pair programming [10], and project-based work, to enhance students' motivation, engagement, and learning outcomes [12]. Among these, Peer Instruction (PI) has gained significant traction as an effective strategy where students individually respond to conceptual questions, discuss with peers, and then revise their answers, leading to improved failure rates, retention, and exam performance in Computer Science [15]. Several in-class coding exercise tools has developed to assist instructors in facilitating collaborative learning in programming classrooms. For example, PuzzleMe [18] enabled real-time peer discussions, balancing code and expertise diversity among students. VizPI [13] supports in-class Peer Instruction by allowing instructors to distribute coding exercises during lectures for synchronous student participation. Große and Renkl's work has shown that grouping students with different solutions has been effective in improving learning outcomes in mathematics [5]. However, the effectiveness of these discussions and different group formations on student coding performance remain areas of active research.

In this paper, we report our experience in conducting a series of collaborative learning sessions at our institute. Our analysis focuses on three key areas: 1) the effects of group discussions on students' coding performance, 2) the impact of different grouping mechanisms (random and expertise-balanced) on discussion activity and student coding performance, and 3) the influence of group composition (varying expertise levels) on discussion quality and student performance. This experiment involved a large-scale study, encompassing 788 student participants engaged in collaborative learning activities over 176 minutes across multiple sessions. The study generated extensive data, including interactions within 255 formed groups, analysis of group discussions, and evaluation of numerous coding submissions.

Our experiment yielded several intriguing results that challenge common assumptions about collaborative learning in programming education. We found that active group participation did not consistently lead to improved coding performance for struggling students.

The presence of expert students (those who achieved a 100% pass rate on programming tasks prior to group formation), while often beneficial, did not guarantee positive outcomes for all group members. In fact, we observed instances where expert-led discussions failed to result in significant improvements for lower-performing students. Furthermore, our study revealed that the grouping method (expertise-balanced or random) had no significant impact on discussion activeness or overall student improvement across sessions. This finding suggests that the effectiveness of collaborative learning may depend more on the quality of interactions and individual student factors than on predetermined group composition strategies. Our experience also uncovered several challenges in implementing collaborative learning in large CS1 classes: the difficulty of achieving ideal expertise-balanced groups in real classroom settings, the tendency of some students to prioritize individual coding over group discussion, and the varying ability of expert students to effectively communicate with their peers. Interestingly, we noted that less active participants sometimes benefited from "lurking" in group discussions, suggesting that silent participation can also be a valid form of learning in these environments. These insights provide valuable lessons for future research on grouping mechanisms and in-class activity design.

2 RELATED WORK

Collaborative learning in computer science education has demonstrated promising results in improving student engagement, performance, and retention [1]. Studies have shown increased student confidence, participation, and proficiency in introductory programming courses [4]. Zingaro et al. [21] investigated the impact of peer discussions on student performance in CS courses and found that such discussions can lead to significant learning gains. However, the quality and nature of peer feedback also varies based on student performance levels. Studies have found that higher-performing students tend to provide longer, more detailed feedback [22], while the benefits of receiving feedback can vary dramatically depending on both the provider's and receiver's performance levels [2]. James et al. [7] observed that over one-third of in-class peer discussions in large classes can be unproductive, highlighting the need for careful design of collaborative activities and group formation.

Group formation in CS classes is crucial for promoting effective collaborative learning. Deibel [3] found that instructor-selected teams for in-class group work can enhance student interaction and learning. Automated group formation mechanisms have shown promise in creating consistent and successful groups for learning activities [20]. However, forming productive groups remains challenging due to the complexity of the problem and the sparsity of the solution space [6]. To address this, researchers have proposed various approaches, including grouping by learning style, latent jigsaw methods [3], and massively parallel brute-force algorithms [6]. Web-based applications that collect student information and use algorithms to form groups offer a more systematic approach to group formation [6]. Nevertheless, the effectiveness of collaborative activities in technical domains such as logic programming remains uncertain.

To support collaborative programming, researchers have developed various tools and platforms [19]. VizGroup facilitates visual collaboration for distributed groups [14]. Lee et al. explored coordination models for ad hoc programming teams [8]. Wang et al. [18] developed PuzzleMe, a tool that enables real-time peer discussions for programming learners while balancing code and expertise diversity. These tools have facilitated the distribution of exercises, grouping of students into small teams, and real-time monitoring of student performance.

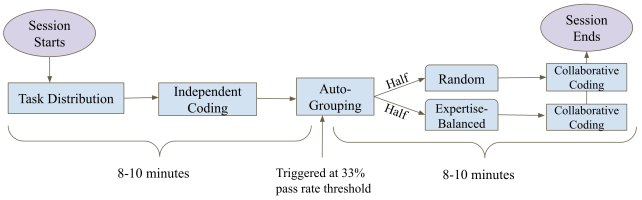

Figure 1: High-level experimental workflow

3 EXPERIMENT SETUP

3.1 CS1 Class

Our experiment was conducted at the author's institution. We chose the Introduction to Programming in Python course during the Spring 2024 semester. With a total enrollment of 694 students spread across three sessions, this course is co-taught by two instructors and supported by a team of 21 TAs. The course is structured entirely as an in-person experience. The curriculum covers a wide range of fundamental programming concepts, including basic control flow with loops and conditionals, state tracing and manipulation, simple and complex data types, functional and object-oriented coding strategies, and data processing. Our experiment was conducted across 11 sessions as detailed in Table 1, leveraging the teaching schedules of both instructors. The course's structure allowed us to seamlessly implement our group discussion and peer instruction activities within the existing framework, which already included regular in-class activities.

3.2 In-class Exercises

Figure 1 illustrates the procedure of the in-class experiments. All experiments were conducted in the CS1 course. Before starting each experiment, an instructor announced the procedure of the peer instruction session and distributed the programming exercise to all students. Students then worked independently on the exercise until the overall class performance reached a 33% pass rate. At this point, students were automatically divided into groups using two different mechanisms: half were grouped randomly, and the other half were grouped based on an expertise-balanced mechanism (see Sec 3.4). During the collaborative coding phase, students in each group shared a text-based chat channel where they could see each other's pass rates in real time. The session concluded when the instructor manually terminated the chat channels. Both the independent and collaborative coding parts lasted approximately 8-10 minutes each. We collected students' chat and code submission data during the experiment sessions, ensuring that no identifying information was included.

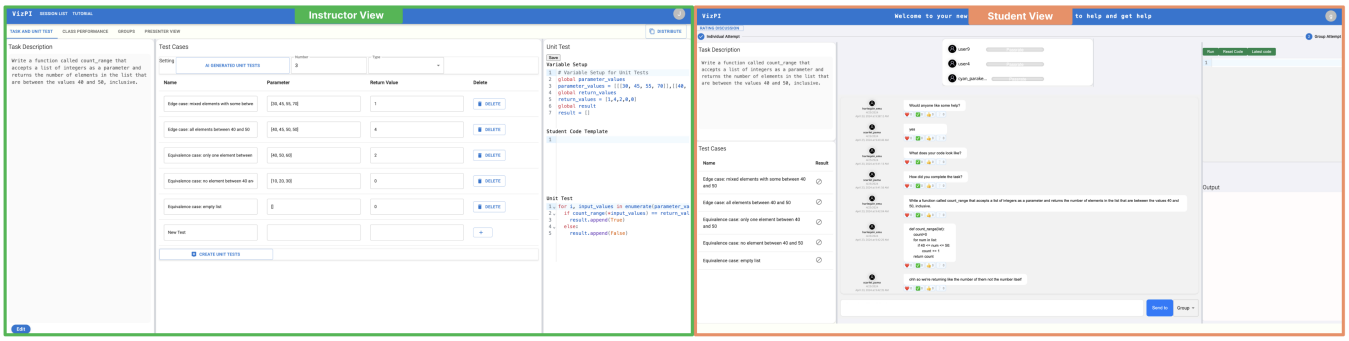

Figure 2: VizPI interface showing the instructor view (left) for managing exercises and group formation, and the student view (right) featuring an integrated IDE and group chat window for collaborative programming

All the programming tasks in the experiment sessions were designed by the instructors of the course, matching students' knowledge levels and the course progress. Several learning objectives such as *Variables*, *Functions*, *Strings*, *Lists*, and *Statements (e.g., if, for, while)* were covered in the tasks.

3.3 Data Collection System

We used a learning platform VizPI [13] to collect student discussion and coding data. The interface (see Figure 2) consists of an instructor UI and a student UI. The instructor UI enables instructors to easily conduct peer instruction activity by creating and distributing coding exercises and dividing the class into small groups. The student UI features an IDE and group chat window where students can write and test their code while communicating with their peers. In each group, students remain anonymous, and they can view each other's pass rate but not their code.

3.4 Grouping Mechanisms

The system first collects comprehensive data on all participating users, focusing on the number of messages sent and their code acceptance rates. Using the Zigzag Distribution Algorithm [16], the system then divides the students into two groups:

- **Random Group:** Students are randomly assigned into groups of three or four.
- **Expertise-balanced Group:** Students are categorized into high, medium, and low performers based on their pass rate.

The system further analyzes the similarity of students' code submissions using the text-embedding-3-large model[1] to generate embeddings, then compares these embeddings using cosine similarity. To maximize within-group code similarity, each group is intentionally formed with one student from each performance category (high, medium, low), ensuring a balanced mix of skills and perspectives. Any remaining students are grouped to ensure no one is left out.

4 RESULTS & LESSONS LEARNED

4.1 Basic Stats & Data Cleaning

We collected data from 11 sessions of CS1 Class, as detailed in Table 1. These sessions involved a total of 862 students, of which 788 students fully participated and 74 were excluded due to late arrivals or being ungrouped. Individual session participation ranged from 52 to 84 students. We formed 255 groups, split evenly between expertise-balanced (128 groups) and random assignment (127 groups) mechanisms. Within both grouping mechanisms, we further categorized groups based on their composition, focusing primarily on the presence of expert students (those achieving a 100% pass rate before grouping). Our improvement analysis focused on these poor-performing students (Pass Rate < 100%). The experiment generated rich interaction data:

- 598 messages exchanged among participants
- Session durations ranging from 7:26 to 29:50 minutes (average ~16 minutes)
- Discussion phases lasting 2:55 to 14:11 minutes (average ~7 minutes)

Our data cleaning process involved filtering messages and normalizing time data to ensure comparability across sessions. This comprehensive dataset and our categorization of group expertise levels provide a foundation for analyzing group discussion effectiveness and the impact of different grouping mechanisms on student performance.

4.2 Group and Discussion Definitions

We established several key definitions and categorized groups accordingly to analyze group dynamics and their impact on student coding performance.

- **Active groups** Groups where messages were exchanged
- **Inactive groups** Groups with no message exchanges
- **Relevant discussion** Instances where at least two students exchanged messages discussing code issues or exercise approaches
- **Irrelevant discussion** Groups with only one message, or where messages were limited to default greetings without substantive replies about the coding task
- **Improvement** Poor-performing students increased their pass rate from pre-grouping to the end of the session
- **No improvement** No increase in pass rate for poor-performing students
- **Have expert** Groups with at least one expert student
- **No expert** Groups without any expert students
- **Expert-led discussion** Discussions led by expert students

Figure 3 illustrates the breakdown of our 255 groups based on these definitions.

[1] https://platform.openai.com/docs/models/embeddings

Session ID	Student total number	Balanced group number	Students in balanced group	Random group number	Students in random group	Session start time	Grouping time	Session end time	Session lasting time	Discussion lasting time
1	52	8	26	8	26	2024-04-22 09:02:42	09:15:47	09:22:46	0:20:04	0:06:59
2	58	9	29	9	29	2024-04-22 09:23:20	09:42:16	09:45:11	0:21:51	0:02:55
3	78	13	39	13	39	2024-04-22 12:32:27	12:34:20	12:39:53	0:07:26	0:05:33
4	84	14	42	14	42	2024-04-22 12:46:07	12:50:16	12:53:59	0:07:52	0:03:43
5	82	13	41	13	41	2024-04-23 09:34:54	09:38:13	09:51:45	0:16:51	0:13:32
6	81	13	41	13	40	2024-04-24 12:26:32	12:34:16	12:37:58	0:11:26	0:03:42
7	62	10	31	10	31	2024-04-24 09:26:54	09:40:01	09:43:02	0:16:08	0:03:01
8	63	10	32	10	31	2024-04-24 09:39:45	09:50:53	09:55:42	0:15:57	0:04:49
9	67	11	34	11	33	2024-04-24 09:00:52	09:16:31	09:30:42	0:29:50	0:14:11
10	83	14	42	13	41	2024-04-24 12:46:12	12:49:59	13:00:05	0:13:53	0:10:06
11	78	13	39	13	39	2024-04-24 13:00:59	13:02:59	13:15:50	0:14:51	0:12:51

Table 1: Each session's Student Number, Grouping Number, and Time Frame Data

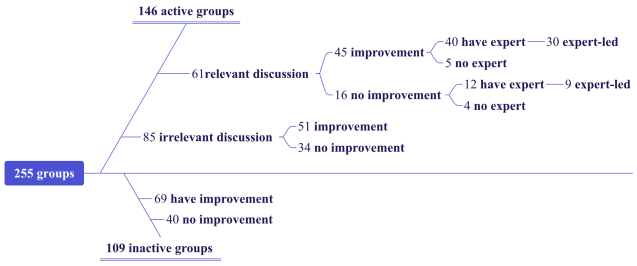

Figure 3: Groups activity and improvement

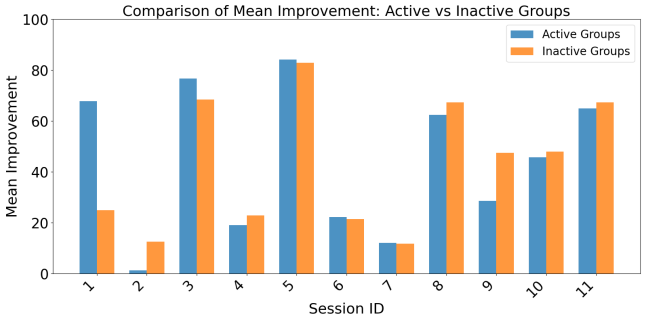

Figure 4: Active and improvement

4.3 Impact of Active and Relevant Group Discussions on Students' Coding Performance

Our analysis revealed that active and relevant discussions generally correlate with improved coding performance, but this relationship is not consistent across all sessions. The effectiveness of group discussions varies widely, with improvement rates ranging from 0% to 85% across sessions. Interestingly, we observed that inactive or irrelevant discussions sometimes led to better performance, highlighting the complexity of collaborative learning in programming education.

4.3.1 Active vs. Inactive Groups. Figure 4 illustrates the breakdown of active and inactive groups and their impact on student improvement. Among the active groups, 96 (66%) showed improvement in poor-performing students' coding skills, compared to 69 (63%) in inactive groups. However, the session-by-session analysis revealed inconsistencies:

- In 5 out of 11 sessions (45%), active groups showed higher mean improvement than inactive groups.
- In 6 sessions (55%), inactive groups outperformed active groups.
- The magnitude of difference between active and inactive group performance varied widely across sessions.

4.3.2 Relevant vs. Irrelevant Discussions. To explore the quality of discussions, we examined the relevance of discussions within active groups (see Figure 5). Of the groups with relevant discussions, 45 (74%) showed improvement, compared to 51 (60%) of groups with irrelevant discussions. This analysis provided additional insights:

- In 4 out of 10 sessions (40%), groups with relevant discussions showed higher mean improvement than those with irrelevant discussions. (Session 6 has 0 relevant discussions, while Session 8 had no active discussions and was excluded from this analysis)
- In 6 sessions (60%), groups with irrelevant discussions unexpectedly outperformed those with relevant discussions.
- The impact of discussion relevance on performance improvement varied considerably across sessions.

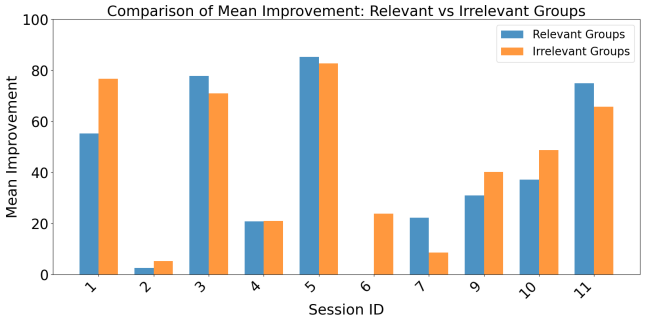

Figure 5: Relevant and improvement

4.3.3 Implications of Findings and Lessons Learned. Our observations suggest that the effectiveness of group discussions may be influenced by multiple factors beyond mere participation or perceived relevance. While not conclusively proven in this study, factors such as task difficulty, student expertise levels, and other contextual elements likely play significant roles. Further research is needed to quantify these effects. Besides, the wide variation in improvement rates across sessions underscores the highly context-dependent nature of group discussions' impact. This observation teaches us that a

one-size-fits-all approach to collaborative learning in programming education is insufficient. Future experiments should incorporate more detailed contextual data to better understand these variations. Perhaps most notably, the performance of some inactive or irrelevant discussion groups highlights the multifaceted nature of learning in programming education. Their performance proves students' independent trial-error process could lead to performance improvement. This finding suggests that our current metrics for categorizing discussions as "active" or "relevant" may not fully capture the nuanced ways in which students learn from peer interactions. It also indicates that silent participation or even seemingly off-topic discussions might have unexpected benefits that our current measurement tools fail to capture.

4.4 The effectiveness of different grouping methods (random vs. expertise-balanced) on discussion activity and student coding performance

Our experiment compared two grouping mechanisms: random and expertise-balanced grouping. We analyzed their impact on group discussion activity and student coding performance across 11 sessions (see Figure 6). The results reveal that there's no clear superiority of one method over the other.

4.4.1 Discussion Activity. The data shows considerable variation in active group ratios (the proportion of groups that engaged in discussion during a session) between sessions and grouping methods:

- Random grouping: Active group ratios ranged from 0% to 93%, with a mean of 55%.
- Expertise-balanced grouping: Active group ratios ranged from 41% to 100%, with a mean of 60%.

While the expertise-balanced groups showed slightly higher average active group ratios, the difference was not consistent across all sessions. Some sessions (e.g., 1, 5) saw higher activity in expertise-balanced groups, while others (e.g., 2, 4) showed higher activity in randomly formed groups.

4.4.2 Student Coding Performance. We measured student coding performance improvement for both grouping methods:

- Random grouping: Average improvement ranged from 9% to 79%, with an overall mean of 46%.
- Expertise-balanced grouping: Average improvement ranged from 0% to 89%, with an overall mean of 44%.

The overall average improvement was slightly higher for random grouping (46%) compared to expertise-balanced grouping (44%). However, this difference was not statistically significant, as indicated by the Wilcoxon signed-rank test (statistic = 29.00, p-value = 0.7646). This suggests that the grouping method (random vs. expertise-balanced) does not significantly impact student improvement.

4.4.3 Session-by-Session Analysis. The effectiveness of each grouping method varied considerably across sessions:

- In 6 out of 11 sessions, random grouping showed higher average improvement.
- In 5 out of 11 sessions, expertise-balanced grouping showed higher average improvement.
- The magnitude of difference in improvement between the two methods ranged from 0.2% (session 10) to 19% (session 11).

Figure 6: Comparison of Random and Expertise-balanced Group Formation Methods Across Multiple Sessions

Notably, some sessions (e.g., 2) showed a stark contrast, with random grouping yielding an 9% improvement while expertise-balanced grouping resulted in no improvement.

4.4.4 Implications and Lessons Learned. Our findings on the effectiveness of random versus expertise-balanced grouping mechanisms reveal a more nuanced picture than initially anticipated. The lack of a clear, consistent advantage for either method across all sessions highlights the complexity of collaborative learning in programming education. This complexity suggests that the effectiveness of grouping strategies may be influenced by a multitude of factors beyond the simple dichotomy of random versus balanced expertise. The varying degrees of success for both methods across different sessions indicate that contextual elements such as task difficulty, student preparedness, and even subtle variations in group dynamics play crucial roles in determining outcomes.

These observations lead us to consider that the apparent similarity in performance between random and expertise-balanced groups might be due to underlying similarities in group composition that occur naturally even in random grouping. With 61 out of 127 random groups having similar expertise levels to the expertise-balanced groups, we must question whether our current approach to expertise-balanced grouping is sufficiently distinct from random assignment to produce measurable differences in outcomes. This realization points to the need for a more granular analysis of group composition and its impact on both group activity and student performance.

4.5 The influence of group composition (varying expertise levels) on discussion activity and student coding performance

In this section, we delve deeper into the detailed group composition to examine whether the lack of significant differences between random and expertise-balanced groups is due to their similar expertise levels. While quantitative analysis across sessions did not yield consistent or statistically significant results, a deeper semantic analysis of group interactions provided valuable insights into the role of expertise in group discussions and its influence on student performance.

4.5.1 Expert-led Discussions and Performance Improvement. A notable trend emerged from our analysis of active discussion groups where poor-performing students showed improvement. Out of 40 such groups, 30 (75%) were expert-led, meaning they included at least one expert student who actively participated in the discussion. This suggests that the presence of an expert can often catalyze productive discussions and lead to performance improvements for struggling students.

For instance, in group 2889 from session 3, we observed a beneficial interaction between an expert student (Student A) and a struggling student (Student B):

> Student A: Would anyone like some help?
> Student B: Yes, I tried to do a list comprehension like this "def good_scores(scores): new_scores = [score for scores in good_scores if score > 80] return new_scores" should I do a for loop instead?
> Student A: I used a for loop, but a comprehension should work. You need to change good_scores to just scores since scores is the list.
> Student B: ohh i fixed it
> Student A: nice :)
> Student B: thanks pink_primate!

This exchange demonstrates how an expert's presence can facilitate a focused, problem-solving dialogue, leading to immediate improvements in the struggling student's code.

4.5.2 Limitations and Factors Influencing the Effectiveness of Expert Presence. Our analysis also revealed that the mere presence of expert students does not guarantee successful knowledge transfer or performance improvement for struggling peers. Here is an example between two expert students (C and E) and one struggling student (Student D) from group 2860 in session 2:

> Student C: For those that didn't get it, please send in a copy of your code
> Student D: "def remove_vowels(phrase): vowels = ('A', 'E', 'I', 'O', 'U') for letter in phrase: if letter in vowels.upper(): phrase.strip(letter) elif letter in vowels.lower(): phrase.strip(letter) return phrase"
> Student E: What errors are you getting?
> Student C: I don't think you can strip a letter from the mddle of the word, only the ends
> Student D: tuple' object has to attribute 'upper'
> Student C: Write your values as a list and not as a tuple
> Student C: Like this maybe: vowels = 'aeiouAEIOU'
> Student E: that's a string, a list is contained in []

In this instance, the struggling student failed to improve despite iterative attempts and group support. Our analysis across 40 groups identified several factors contributing to the varying effectiveness of these interactions: insufficient problem identification, fragmented or ambiguous advice, temporal constraints, conflicting guidance from multiple sources, and the absence of structured, step-by-step problem-solving methodologies. These factors were observed repeatedly in discussions where expert-led interactions failed to yield significant progress for struggling students.

These findings underscore the challenges faced by expert students when attempting to contribute effectively to group discussions. The efficacy of expert-led discussions is contingent upon the expert's ability to communicate clearly, identify core issues, and provide structured, tailored guidance to struggling peers. This observation elucidates why expertise-balanced group mechanisms do not consistently yield significant improvements in student performance. The key inference is that leveraging expertise within group discussions is a multifaceted challenge, emphasizing the necessity for targeted strategies to enhance peer instruction in collaborative coding environments. Future research should focus on developing and evaluating such strategies to optimize the benefits of Peer Instruction learning in computer science education.

5 DISCUSSION & CONCLUSION

Our main observations from this experiment including: 1) Active discussions generally correlated with improved coding performance, but this relationship was not consistent across all sessions. The effectiveness of group discussions varied widely, with improvement rates ranging from 0% to 85.22% across sessions. 2) Different grouping method (expertise-balanced or random) does not significantly influence group discussion activeness and students' improvement. And 3) 75% of productive groups (groups have active discussion and improved performance) were led by expert students. This finding aligns with Vygotsky's concept of the Zone of Proximal Development [17]. Expert students often provided scaffolding for their struggling peers, facilitating knowledge transfer and problem-solving skills. However, the mere presence of an expert did not guarantee successful outcomes. Factors such as communication clarity, problem-identification skills, and the ability to provide structured guidance influenced the effectiveness of expert-led discussions. This complexity echoes findings by James and Willoughby [7], who observed that a significant portion of in-class peer discussions can be unproductive.

Our experiment highlighted several challenges in implementing effective collaborative learning strategies in large CS1 courses. Firstly, achieving an ideal expertise balance in real classroom settings proved to be a complex task, often deviating from theoretical expectations. Secondly, we observed a tendency among some students to prioritize individual coding efforts over engaging in group discussions, potentially limiting the benefits of collaborative learning. Lastly, the effectiveness of grouping mechanisms, whether random or expertise-balanced, exhibited considerable variation across sessions. This variability suggests that uniform approaches to group formation may be insufficient in addressing the diverse learning needs and dynamics present in large-scale programming courses.

REFERENCES

[1] Leland Beck and Alexander Chizhik. 2013. Cooperative learning instructional methods for CS1: Design, implementation, and evaluation. *ACM Transactions on Computing Education (TOCE)* 13, 3 (2013), 1–21.

[2] Catherine H Crouch, Jessica Watkins, Adam P Fagen, and Eric Mazur. 2007. Peer instruction: Engaging students one-on-one, all at once. *Research-based reform of university physics* 1, 1 (2007), 40–95.

[3] Katherine Deibel. 2005. Team formation methods for increasing interaction during in-class group work. In *Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education*. 291–295.

[4] Katrina Falkner and David S Munro. 2009. Easing the transition: a collaborative learning approach. In *Proceedings of the Eleventh Australasian Conference on Computing Education-Volume 95*. 65–74.

[5] Cornelia S Große and Alexander Renkl. 2006. Effects of multiple solution methods in mathematics learning. *Learning and Instruction* 16, 2 (2006), 122–138.

[6] Tyson R Henry. 2013. Forming productive student groups using a massively parallel brute-force algorithm. In *Proceedings of the World Congress on Engineering and Computer Science*, Vol. 1. 23–25.

[7] Mary C James and Shannon Willoughby. 2011. Listening to student conversations during clicker questions: What you have not heard might surprise you! *American Journal of Physics* 79, 1 (2011), 123–132.

[8] Sang Won Lee, Yan Chen, Noah Klugman, Sai R Gouravajhala, Angela Chen, and Walter S Lasecki. 2017. Exploring coordination models for ad hoc programming teams. In *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. 2738–2745.

[9] Leo Porter, Cynthia Bailey Lee, and Beth Simon. 2013. Halving fail rates using peer instruction: a study of four computer science courses. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education* (Denver, Colorado, USA) *(SIGCSE '13)*. Association for Computing Machinery, New York, NY, USA, 177–182. https://doi.org/10.1145/2445196.2445250

[10] Fernando J. Rodríguez, Kimberly Michelle Price, and Kristy Elizabeth Boyer. 2017. Exploring the Pair Programming Process: Characteristics of Effective Collaboration. In *Proceedings of the 2017 ACM SIGCSE Technical Symposium on Computer Science Education* (Seattle, Washington, USA) *(SIGCSE '17)*. Association for Computing Machinery, New York, NY, USA, 507–512. https://doi.org/10.1145/3017680.3017748

[11] Roberta Evans Sabin and Edward P Sabin. 1994. Collaborative learning in an introductory computer science course. In *Proceedings of the twenty-fifth SIGCSE symposium on Computer science education*. 304–308.

[12] Luis Miguel Serrano-Cámara, Maximiliano Paredes-Velasco, Carlos-María Alcover, and J Ángel Velazquez-Iturbide. 2014. An evaluation of students' motivation in computer-supported collaborative learning of programming concepts. *Computers in human behavior* 31 (2014), 499–508.

[13] Xiaohang Tang, Xi Chen, Sam Wong, and Yan Chen. 2023. VizPI: A Real-Time Visualization Tool for Enhancing Peer Instruction in Large-Scale Programming Lectures. In *Adjunct Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology* (San Francisco, CA, USA) *(UIST '23 Adjunct)*. Association for Computing Machinery, New York, NY, USA, Article 17, 3 pages. https://doi.org/10.1145/3586182.3616632

[14] Xiaohang Tang, Sam Wong, Kevin Pu, Xi Chen, Yalong Yang, and Yan Chen. 2024. VizGroup: An AI-assisted Event-driven System for Collaborative Programming Learning Analytics. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology* (Pittsburgh, PA, USA) *(UIST '24)*. Association for Computing Machinery, New York, NY, USA, Article 93, 22 pages. https://doi.org/10.1145/3654777.3676347

[15] Cynthia Taylor, Jaime Spacco, David P Bunde, Andrew Petersen, Soohyun Nam Liao, and Leo Porter. 2018. A multi-institution exploration of peer instruction in practice. In *Proceedings of the 23rd annual ACM conference on innovation and technology in computer science education*. 308–313.

[16] Duc A Tran, Kien A Hua, and Tai T Do. 2004. A peer-to-peer architecture for media streaming. *IEEE journal on Selected Areas in Communications* 22, 1 (2004), 121–133.

[17] Lev Semenovich Vygotsky and Michael Cole. 1978. *Mind in society: Development of higher psychological processes*. Harvard university press.

[18] April Yi Wang, Yan Chen, John Joon Young Chung, Christopher Brooks, and Steve Oney. 2021. PuzzleMe: Leveraging Peer Assessment for In-Class Programming Exercises. *Proceedings of the ACM on Human-Computer Interaction* 5, CSCW2 (2021), 1–24.

[19] Tianjia Wang, Tong Wu, Huayi Liu, Chris Brown, and Yan Chen. 2024. Generative Co-Learners: Enhancing Cognitive and Social Presence of Students in Asynchronous Learning with Generative AI. *arXiv preprint arXiv:2410.04365* (2024).

[20] Carine G Webber and Maria de Fátima Webber do Prado Lima. 2012. Evaluating automatic group formation mechanisms to promote collaborative learning–a case study. *International Journal of Learning Technology* 7, 3 (2012), 261–276.

[21] Daniel Zingaro and Leo Porter. 2014. Peer instruction in computing: The value of instructor intervention. *Computers & Education* 71 (2014), 87–96.

[22] Zheng Zong and Christian D Schunn. 2023. Does matching peers at finer-grained levels of prior performance enhance gains in task performance from peer review? *International Journal of Computer-Supported Collaborative Learning* 18, 3 (2023), 425–456.