Exploring Differences in Planning between Students with and without Prior Experience in Programming

Ryan Parsons, Western Washington University

Ryan Parsons has taught introductory Computer Science for 6 years at Whatcom Community College. He served as the Program Coordinator for the newly created Software Development program there. He has been working on his Master's in Computer Science at Western Washington University, where his research focus has been on Computer Science Education.

Qiang Hao, Western Washington University

Associate professor of computer science

Dr. Lu Ding, University of South Alabama

Dr. Lu Ding researches critical aspects of teaching and learning in STEM fields such as student engagement and motivation in online learning environments. Currently, Lu's research involves how to design instructional videos for teaching debugging skills and misconceptions in learning programming. Another research line of hers is gaming and game-based learning (GBL), especially in empowering teachers to use GBL in their everyday teaching and to engage students in learning.

Exploring Differences in Planning Between Students With and Without Prior Experience in Programming

Ryan Parsons
parsonr6@wwu.edu
Western Washington University

Qiang Hao
qiang.hao@wwu.edu
Western Washington University

Lu Ding
luding@southalabama.edu
University of South Alabama

Abstract

Planning, as a metacognitive skill, is critical to problem solving in computer science. Within the context of computer science, planning constitutes of understanding the problem by reading and exploring the prompt, breaking the problem down into smaller pieces such as listing methods and using pseudocode, and determining what tools are needed to solve the problem. Understanding the fine-grained differences between students with and without prior programming experience in terms of planning can help educators provide better guidance to novice learners in terms of how to approach problem-solving and design solutions given a programming task. Prior studies have contributed to understanding different aspects of problem solving in computing education, such as digesting errors, and debugging. However, few studies focused on the step of planning that happens mainly prior to problem solving. To fill this gap, we studied 48 students with and without prior programming experience in terms of planning prior to working on a programming assignment in the context of a second introductory programming course (CS2) via surveys and follow-up interviews. Our results shed light into the fine-grained differences in planning between novice and experienced learners. We discussed these differences and how they can be used to guide meaningful interventions that focus on megacognitive skills in computing education.

1 Introduction

A deep understanding of the difference between students who had experience in programming and those who had no prior experience can help instructors make informed decisions in how to teach both groups in the same course more effectively. As more students are exposed to programming in high school, it is increasingly likely for an introductory programming course to have both students who had experience in programming and those who had no prior experience. Undoubtedly, students with no prior experience in programming struggle more than their counterparts in such courses [1]. The understanding of how students with no prior experience

approach different challenges in coursework can contribute to insights into how to help students with no experience more effectively.

In an effort to better support students with no prior experience in programming in introductory computer science courses, prior studies have studied how prior experience in programming contributes to the difference in problem solving and dealing with code errors, self-regulation, and metacognitive awareness [2], [3], [4], [5], [6]. These studies contributed to a better understanding of how students with no prior experience in programming think and approach problem-solving. However, few studies have explored how students with no prior experience plan for programming assignments, and how they differ from their counterparts who have exposure to programming.

In the context of programming courses, planning consists of decomposing the problem into smaller, manageable pieces, and is the problem solving stage of understanding the problem and devising a plan [7]. Understanding the problem might take the form of reading through the assignment prompt, identifying the goal of the programming problem, exploring the problem, among others. Devising a plan might include strategies such as breaking down the problem through listing the methods, writing pseudocode to develop a general algorithm, determining what tools are needed to solve the problem, etc. Effective planning for programming assignments leads to better success on these assignment, which in turn leads to more confidence and success in the field of computer science. Often in introductory computer science courses, the onus is entirely on the students to perform some sort of planning before beginning to write their code, requiring self-regulated learning to be successful. Students with prior exposure to programming, drawing on their prior experience, may have developed certain strategies to help them in this area of planning that novice programmers may not utilize due to being unaware of these strategies or their usefulness.

This paper aims to fill the gap in the literature by exploring how students without prior experience programming plan for programming assignments, and how they differ from their counterparts who had experience in programming in the context of introductory programming courses. In particular, this paper will explore the following research question:

- How do learners with no prior programming experience differ from those with prior programming experience in planning for and decomposing programming assignments?

2 Related Work

Problem solving is a key skill in computer science. In order to better support students in introductory computer science courses, educators and educational researchers need to understand how students approach problem solving. As defined by Polya, problem solving has four distinct principles: understand the problem, devise a plan, carry out the plan, and look back [7]. It is important to understand how students interact with each of these four principles. Barnes et al. describe these principles as they specifically relate to computer science [8]. Understanding the problem consists of reading through the prompt and questioning it to ensure they have a full understanding of what is being asked of them. Additionally, analyzing any provided sample inputs and outputs is part of this step. Finding related or similar problems to base their new solution on, breaking the problem down into classes and methods, and writing pseudocode are

part of the devising a plan principle. Carrying out the plan is when one writes the actual code, turning the devised plan into an actual solution. This includes the iterative process of debugging and fixing errors in the code. Finally, looking back or reviewing is when one reflects on the final product, thinking metacognitively about the entire process to improve upon the steps taken for future problems.

General coding mistakes is one of the large barriers to success for students with no programming experience. Prior studies exploring student problem solving primarily focused on students' coding, debugging, and errors. These studies show that most errors can be categorized into a handful of common errors that students with no prior experience make [9], [10], [11]. Focusing on these errors to find better ways to prevent students from making them is an important endeavor. However, these errors do not solely come from coding itself. While exploring errors of students with no prior programming experience, Ebrahimi et al. specifically looked at plan and plan composition errors using the rainfall problem [2]. They define these ”plans” as general algorithms, and compare students with prior experience in programming to those without noting that students with programming experience are able to draw upon their experience for these ”plans” allowing them to more easily solve some programming problems as opposed to students without programming experience. Furthermore, taking a step back and looking at problem solving not specifically in the field of computer science, Eichmann et al. explored the role of planning in complex problem solving [12]. Using three indicators, duration, delay, and variance, found that planning earlier in a task led to better performance on the task itself in general with easier problems, but not necessarily on more difficult problems. This means that the level of difficulty on programming assignments might change how students plan for those assignments. In information problem solving tasks the led to a writing product, it was found that experienced problem solvers spent more time on the whole task compared to those without prior experience [13]. They also noted that those with prior experience tended to pay more attention to formulating and reformulating the problem than those without prior experience. Through exploring the planning stages of problem solving and comparing students with no programming experience to those with prior programming experience, we can gain better insight into what errors may be occurring from the outset, and how we might be able to prevent those.

Other studies investigating student self-regulation found that planning is an integral part of the self-regulation process, especially for students with no experience in programming. Prior studies have shown that students with no prior programming experience tend to struggle with some of the self-regulated learning activities and planning in particular. Falkner et al. performed a study analyzing student reflections in order to explore novice self-regulated learning strategies [14]. They identified that the understanding the problem part of the planning process is extremely complex and differs student by student, requiring specific scaffolding to hone. In a literature review on novices in computer science, Allwood discovered that novices tend to spend less time in the understanding a problem phase of problem solving compared to more experienced programmers and that novices have a much more difficult time drawing from their limited previous experience on other coding problems to find a solution to their current problem [15]. Looking into how introductory programming students self-regulate in the problem solving process when approaching a programming assignment, Loksa et al. found that specifically teaching problem solving for programming can significantly help students with little programming experience through improving efficiency, promoting independence, increasing self-efficacy gains,

and reinforcing a growth mindset [16]. By taking time to cover the general problem solving stages in class, they reduced the amount of students stuck in the design process that didn't know how to approach solving a problem by 2.3%. Additionally, Loksa and Ko explored how self-regulation impacts student success on programming projects [5]. In their think-alouds, they found that the majority of participants explicitly verbalized planning. Only two of their participants from a CS1 course did not mention planning. They noted that the more participants mentioned planning and comprehension monitoring, the fewer errors the participants tended to have, indicating that having students focus more on planning stages of programming problems can lead to better success on programming assignments. Investigating another strategy, Marguliex et al. found that using subgoal labeled worked examples had a positive impact on programming performance [17]. In an attempt to improve metacognitive awareness around planning and decomposing the problem, some researchers have had success with students specifically reflecting on their prior work and making plans for future improvement and utilized automated assessment tools to help improve students' ability to make sense of programming problem prompts [18], [6].

Additional research has gone in to understanding the habits and strategies used by students without prior programming experience in the problem solving process, finding that there are certain habits and strategies that lead to higher success on programming problems, such as starting assignments early, focusing on the relationships between tasks in a program, and visualizing the solution to the problem. Over the the course of five years, Edwards et al. explored effective and ineffective behaviors of students in their first three programming courses [19]. They found that students that started and finished earlier tended to receive higher grades on their assignments, but that they did not tend to spend additional time on their work compared to other students. Researching how students with no prior programming experience move between high-level tasks and low-level code implementations, Castro et al. found that students tend to focus on core tasks of a programming problem, but can struggle with determining the relationships between these tasks [4]. Begum et al. looked in to what specific activities students without prior programming experience performed to understand a programming problem and design a solution for it, noting positive and negative impacts of each activity [3]. Planning, visualizing the solution was found to have a 67% positive effect on the outcome. These studies focus on general habits and strategies that students without prior programming experience tend to have and use.

While there has been much research around students without prior programming experience and how they approach programming problems, there are still some limitations and gaps in the literature on this topic. First of all, most of the literature explored how these students think in isolation. There are not many that compare how they differ from those with prior programming experience [2], [13]. In particular, more work can go in to comparing students without programming experience and students with programming experience in the same context. The comparison can provide insights into strategies and thought processes that are employed by one group or another that can lead to success or failure, allowing educators to tailor their courses to provide better scaffolding for success of all students, especially novice learners. Additionally, most research into the problem solving process of students without prior programming experience examined the process as whole, focusing more on the coding aspects as opposed to the planning and decomposition portions for a problem. By honing in on the planning stages of the problem solving process, we can better understand how students approach programming problems to begin with and identify strategies that lead to success that students with prior programming experience

may employ. This study aims to fill these gaps by comparing students with and without prior programming experience in terms of the planning phase in problem solving. The differences between these two groups can reveal if there are differences in trends or techniques used by novice and more experienced learners, which can inform how planning, as a metacognitive skill, can be taught in introductory computer science courses.

3 Research Methods

To answer the proposed research question, this study examined the differences in planning between students with and without prior programming experience in their second introductory programming course (CS2) via a survey and a follow-up interviews. The CS2 course covers topics such as abstract data types, generics, recursion, and linear data structures. This research was conducted at a large public university located in the northwestern United States with a large computer science program. Students in a CS2 course were asked to participate in this research.

The survey (see the Appendix) that was distributed is composed of three sections. The first section of the survey collected student demographic information. The second section of the survey was used to identify if a student has sufficient programming experience prior to taking the current course. To achieve this purpose, a combination of prior programming courses and Likert Scale questions were used. These questions allowed participants to self-identify their level of experience in order to separate participants into the two experience groups explored in this study. The Likert scale provided a continuum to mitigate preconceived notions for each end of the scale. Participants who reported taking 3 prior programming courses before their current CS2 course and somewhat or strongly agreed with the statement *I have significant prior programming experience.* were identified as students with sufficient prior programming experience. Participants who reported taking 0 or 1 programming course and somewhat or strongly agreed with the statement *Overall, I consider myself a novice programmer.* were identified as students with little to no programming experience.

The third section of the survey asked participants open-ended questions to detail their approach to planning for a programming problem and the process they use to decompose a programming problem. Specifically, participants were asked to reflect on a recent programming assignment and responded to the following prompt: *Please describe your planning process used with as much detail as possible.* In these responses, participants detailed their strategies as they pertained to their planning process in general. Narrowing the scope to problem decomposition in particular, participants, still reflecting on a recent programming assignment, were asked to respond to the following prompt: *Please describe how you decompose/break down the problem for a typical assignment with as much detail as possible.* Further questions were asked about how much time participants spent on planning for a typical programming assignment and how much time overall participants spent on a typical programming assignment.

To further our understanding of the planning phase of students, we designed a 30 minute follow-up interview that uses a recent programming assignment as an example to solicit more detailed information in terms of the planning phase, which can not be easily captured via the survey. During the interview, we also asked participants to reflect on how their planning practices

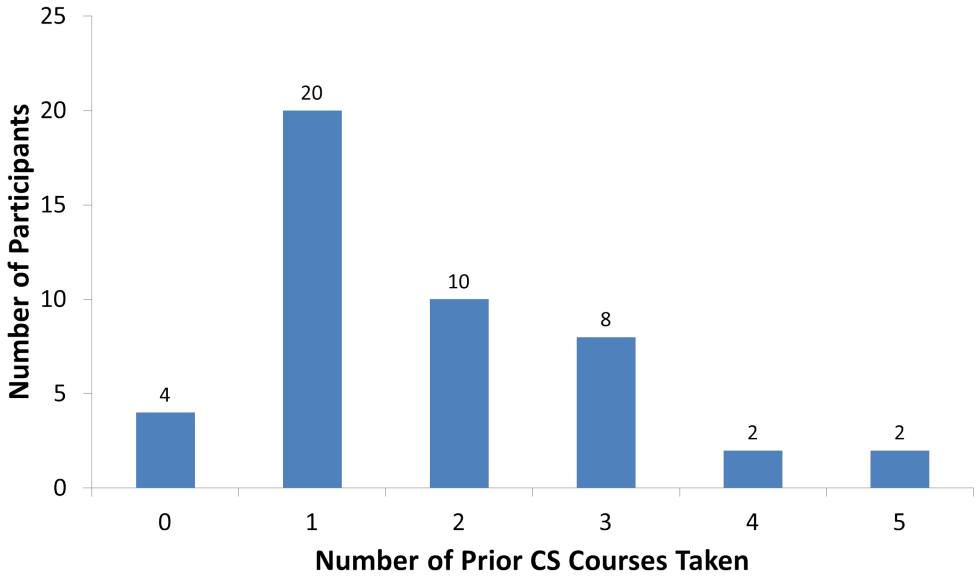

Figure 1: Prior CS courses taken by participants

have evolved over time as they gained more programming experience. Two students participated in the follow-up interview, one without prior programming experience and the one with prior programming experience. Interview questions used for this can be found in Appendix.

To analyze the qualitative data received in the online survey, we utilized the grounded theory for analysis of the collected data. Grounded theory, in contrast to other qualitative data analysis techniques, utilizes the data collected to create a coding framework instead of starting off with a framework to begin with [20]. This allows us to iteratively code our data. We used two stages for this process. The first stage was an open coding where the we broke the data down by concepts based on the qualitative data provided, using as many codes as needed to describe these concepts. This stage yielded 9 codes specifically related to planning and 11 codes related to decomposition. Stage two of the coding process was an axial coding in which we took another pass over our coding framework to refine it using other existing theoretical frameworks and comparisons within the data itself. The second stage resulted in 6 themes for both planning and decomposition. By utilizing grounded theory to analyze this data, we avoid forcing observations into pre-determined, fixed categories and avoid classifying things incorrectly.

4 Results

4.1 Demographics and Prior Programming Experience

Among the 48 participants of this study, the majority were identifying as Caucasian male between 18 and 20 years old. Among all the participants, 38 were identifying as male, 6 identifying as female, and 2 identifying as non-binary. Ethnically, 33 participants stated they were Caucasian, 1 participant stated they were African-American, 3 participants stated they were Latino or Hispanic, 3 participants stated they were Asian, 4 participants stated they were of two or more ethnicities, and 1 participants stated their ethnicity was other or unknown. The majority of participants were

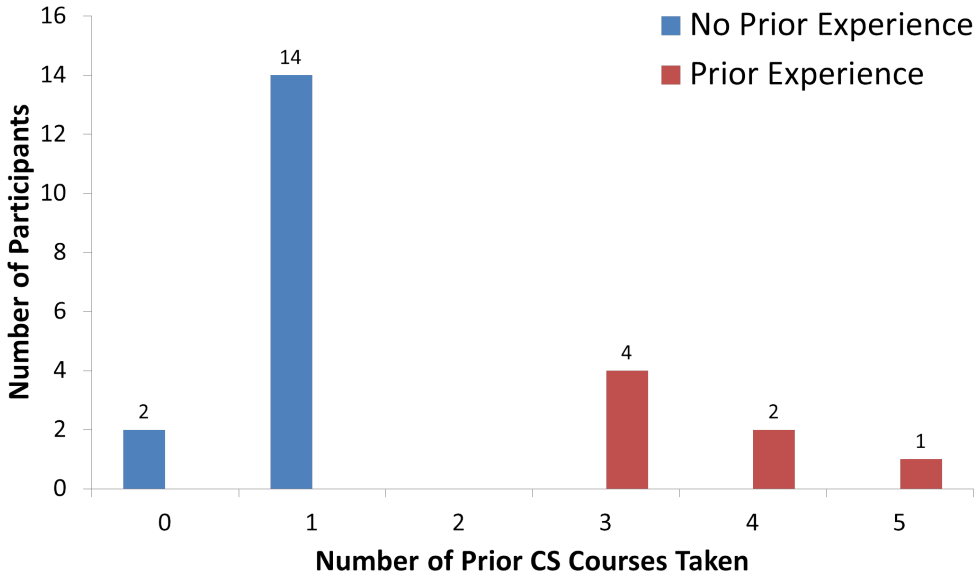

Figure 2: Prior CS courses taken by participants with and without prior programming experience

within the age range of 18 to 20 years old, consisting of 35 participants. 5 of the participants were between 21 and 23 years old, and and additional 5 participants were 24 years old or older.

There was a fairly balanced distribution of students across the various years of university with 15 participants being in their first year, 16 participants being in their second year, 13 students being in their third year, 1 participant being in their fourth year, and 1 participant being in their fifth year. The number of programming courses participants had completed varied from 0 up to 5 (see Figure 1). Most of the participants that were categorized as not having prior programming experience had only taken one prior programming course, the first introductory computer science course (CS1), and the majority of students that had prior programming experience had taken three prior programming courses (see Figure 2).

Most participants (26 out of 48) completed their most recent programming course in the previous quarter. For 4 participants, their current programming courses is their first they have taken. 3 participants took their most recent programming course 2 quarters ago. 4 participants took their most recent programming course 3 quarters ago. 4 participants took their most recent programming course 4 quarters ago. 1 participant took their most recent programming course 5 quarters ago. 4 participants took their most recent programming course more than 6 quarters ago. Using the criteria stated in the Methods section, we identified 7 participants to have sufficient experience in programming, and 16 participants as with little to no prior experience in programming.

4.2 Coding Framework

In the open coding process, we created a total of 9 unique codes for the free-form responses related to planning and 11 unique codes for the free-form responses related to decomposition. Some of the codes generated from the planning prompts included utilizing psuedocode, listing

methods needed, creating an outline, and reading and reflecting on the instructions. For decomposition, some of the codes generated there included reading the assignment, determining the overall goal of the assignment, listing methods needed, and performing research. Both sets of responses were analyzed and coded separately, so there were many codes that overlapped between the codes of each. To further refine our codes into themes, we based the axial coding process on Polya's four stage problem solving method: understanding the problem, devising a plan, carrying out the plan, and looking back [7]. As we focus on the planning and decomposition in this study, we paid particular attention to the first two stages. We further utilized Barnes et al. to specifically apply Polya's problem solving method to programming and Wiedenbeck and Scholtz to break down the first two stages of Polya's method into further detail [8], [21].

As there were many codes that were similar between the planning and the decomposition responses, we were able to come up with one singular framework to utilize for both. The final framework we arrived upon is presented in Tables 1 and 2.

Table 1: Planning Strategies

Strategy	**No Prior Exp Freq**	**No Prior Exp Percent**	**Prior Exp Freq**	**Prior Exp Percent**
Understanding the Problem				
Define the Problem	7	44%	1	14%
Explore the Problem	0	0%	2	29%
Devising a Plan				
List Procedure Names	8	50%	4	57%
Write Pseudocode	10	63%	3	43%
Comment code segments	1	6%	1	14%
Determine libraries/tools	1	6%	1	14%

An individual participant may utilize more than one strategy

Table 2: Decomposition Strategies

Strategy	**No Prior Exp Freq**	**No Prior Exp Percent**	**Prior Exp Freq**	**Prior Exp Percent**
Understanding the Problem				
Define the Problem	6	38%	5	71%
Explore the Problem	3	19%	0	0%
Devising a Plan				
List Procedure Names	2	13%	2	29%
Write Pseudocode	3	19%	2	29%
Comment code segments	0	0%	0	0%
Determine libraries/tools	2	13%	1	14%

An individual participant may utilize more than one strategy

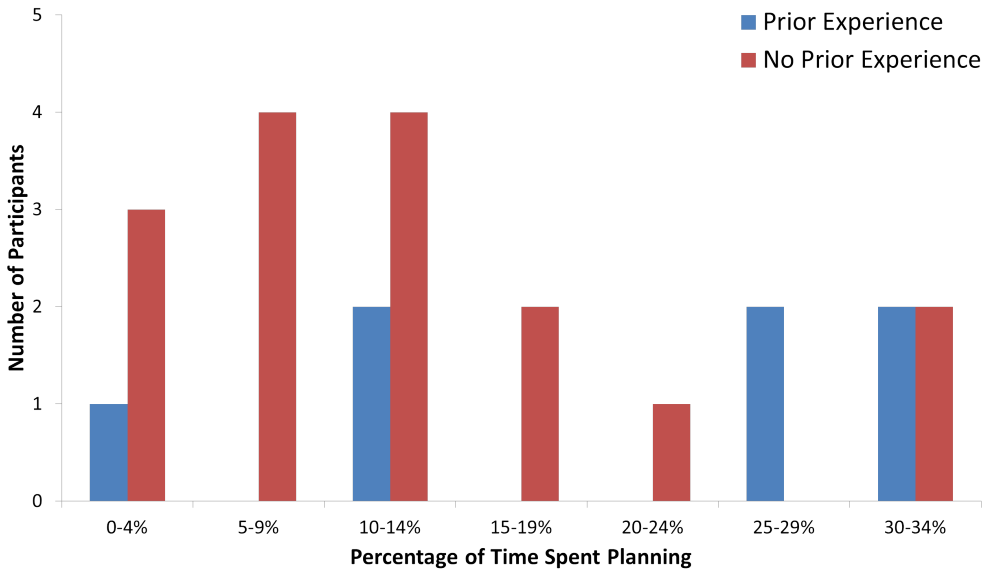

Figure 3: The percentage of time spent on planning by students with and without prior programming experience

4.3 Planning Strategies Results

As we compare the results from Table 1 looking at students with no prior programming experience compared to those with prior programming experience, there are a few interesting observations. First of all, students with no prior experience tended to mention strategies related to understanding the problem much more than students with prior experience. However, all of the focus by students with no prior experience was on defining the problem with no students without prior programming experience mentioning exploring the problem, while 29% of the students with prior experience focused on exploring the problem.

Looking in to the responses from participants, most of the participants that noted a strategy to define the problem specifically mentioned reading through the instructions one or more time to obtain a full understanding of what they were meant to accomplish in this programming problem. The two students with prior experience that mentioned exploring the problem noted strategies that focused on specifics of how to solve the problem instead.

For the devising a plan category, the results between students with and without prior programming experience are pretty similar. It is interesting to see that the most common strategies for both categories of students are to list procedure names and write pseudocode. However, listing the methods one plans to have and creating some general pseudocode for the structure of the program are both commonly taught strategies for planning programming assignments.

The survey also asked participants how much time they typically spent planning for a programming assignment and how much total time they spent on a programming assignment. Using these values, we found the percentage of time spent planning for a programming assignment (see Figure 3). We noticed that students with no prior programming experience tended to spend less time as a percentage of their total time planning compared to students with

prior programming experience. In Figure 3, the students without prior experience tended to cluster between 3% and 17%, while students with prior experience clustered more in the 25% to 33% range.

4.4 Problem Decomposition Results

Comparing the results from Table 2, we see a much larger percentage of students with prior experience defining the problem compared to those without prior experience. The opposite was the case planning. It is possible the that phrasing of ”planning” versus ”decomposition” have different meanings to students with and without experience programming . It could also be that students with prior programming experience focus more on decomposition when planning than those without prior experience.

When decomposing a programming assignment, much of the work is focused on understanding the problem with those categories having the highest frequencies. Devising a plan tended to have similar low frequency results across both those with and without prior experience. However, students with prior experience had slightly higher percentages that used the strategies of listing procedures names or writing pseudocode compared to those without prior experience.

The higher frequencies of understanding the problem strategies compared to devising a plan strategies indicate the both students with and without prior programming experience consider defining the problem and the goals of the programming assignment are the main ways they break programming problems down. They tend to start from the higher level program goals and work down to more specifics. However, many work their way to the specifics decomposing the problem more as they write their code as opposed to at the beginning in the initial planning stages.

4.5 Interview Results

The follow-up interview was conducted with one participant who had no programming experience and the other who had prior programming experience. Both participants likened their planning stages to creating an outline for an essay, coming up with the skeleton of the code before starting to program. The student with prior experience also noted that they often spend time comparing their current problem to problems they have solved before, looking for similarities, which is consistent with the findings of Ebrahimi et al. [2]. This was not something mentioned by the student without prior experience. These results are consistent with Table 1 results, showing that students with prior programming experience explore the problem more than their counterparts. This is likely due to their experience that they can draw upon.

When asked to reflect on their how their planning has changed over time, the student with prior experience noted that they spent more time planning and their planning has become more detailed as they gained more experience. It is possible that part of this is due to comparing the current problem to previous similar problems to see what they can adapt to this current situation.

5 Discussion

By analyzing the survey results, we identified that the largest difference between students with and without prior programming experience as understanding the problem. Interestingly, the

strategies for devising a plan tended to be similar among all the students. These findings indicate the programming problem prompts are analyzed differently by students with and without prior experience, with students with more experience exploring the problem more, actively defining and reconstructing the problem during problem decomposing. Our findings are in line with the findings of prior studies on this aspect [15], [14].

Overall, we found that students without prior experience often had strategies in place for planning despite their lack of experience. These strategies tended to be the common strategies taught in introductory computer science courses of breaking the problem down into methods and writing pseudocode. Students with prior programming experience used the same strategies indicating that these strategies likely are effective as seen in prior work as well [3].

While students with prior experience may not necessarily spend more time on a programming assignment overall as discovered by prior work [19], we found that they were more likely to spend a larger percentage of their overall time on a programming assignment dedicated to planning than students without prior experience. This could be due to more time spent exploring the problem, or possibly that students with prior experience may go into more detail when planning. Performing think-aloud studies to obtain more insight into how students with prior experience and those without spend their time planning is an area that could be explored more in a future study.

This work leads to further questions about programming assignment prompts, such as if programming assignment prompts can be constructed better to help students with little to no prior programming experience. It is important to help students without prior experience develop these planning skills so that they can develop into confident, experienced programmers. Further study could go into exploring how students with and without prior programming experience analyze programming assignment prompts, perhaps through a think-aloud study as those have been shown to be effective ways to understand thought processes [22], to see how programming assignment prompts could be improved.

6 Limitations

There are several limitations of this study. First of all, there is a relatively low number of participants that took part in this study. With more data, there could be other interesting findings. Additionally, replication studies have been shown to be important to obtaining a complete and systematic understanding of issues students without prior programming experience tend to encounter [23]. This is an opportunity for some future work. Another limitation of this study is that the data collected was all self-reported through an online survey. This provides a good overview; however, other data collection methods such as think-alouds [22], more interviews, etc. could be used to obtain more details about the thought processes of students with and without prior programming experience.

There are several areas for future work based on this study. Beyond replicating this study on a larger scale or using other data collection methods, exploring how students with and without prior programming experience differ specifically in how they define and explore programming problems would be an area of further research. Additionally, one interviewee mentioned that they spend more time planning when working with a group on a programming assignment. It would be

interesting to explore how individual work compares to group work when it comes to planning in the context of introductory computer science courses.

7 Conclusion

We explored and compared how students with and without prior programming experience approached planning and decomposing problems in programming assignments in the context of a CS2 course. We identified and categorized common planning and decomposition strategies used by participants, and compared the strategies used by students with and without prior programming experience. Our results shed light into the fine-grained differences in planning between novice and experienced learners. We discussed these differences and how they can be used to guide meaningful interventions that focus on megacognitive skills in computing education. In particular, these results show that CS educators can support students without prior programming experience through focusing on the steps of problems solving, specifically with exploring the problem. Additionally, CS educators can provide tools to help students without prior programming experience with their planning process, helping them to spend more time in the planning phase in order to help them achieve more success while actually coding their solution to the programming problem like we see with students with prior programming experience.

References

[1] C. Watson and F. W. B. Li. Failure rates in introductory programming revisited. In *Proceedings of the 2014 Conference on Innovation &; Technology in Computer Science Education*, ITiCSE '14, page 39–44, New York, NY, USA, 2014. Association for Computing Machinery.

[2] A. Ebrahimi, D. Kopec, and C. Schweikert. Taxonomy of novice programming error patterns with plan, web, and object solutions. *ACM Computing Surveys*, 38(2):1–24, 2006.

[3] M. Begum, J. Nørbjerg, and T. Clemmensen. Novice programming strategies. In *Proceedings of the 2018 SIGED International Conference on Information Systems Education and Research*, pages Paper–5. Association for Information Systems. AIS Electronic Library (AISeL), 2018.

[4] F. E. V. Castro and K. Fisler. Qualitative analyses of movements between task-level and code-level thinking of novice programmers. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, SIGCSE '20, page 487–493, New York, NY, USA, 2020. Association for Computing Machinery.

[5] D. Loksa and A. J. Ko. The role of self-regulation in programming problem solving process and success. In *Proceedings of the 2016 ACM Conference on International Computing Education Research*, ICER '16, page 83–91, New York, NY, USA, 2016. Association for Computing Machinery.

[6] J. Prather, R. Pettit, B. A. Becker, P. Denny, D. Loksa, A. Peters, Z. Albrecht, and K. Masci. First things first: Providing metacognitive scaffolding for interpreting problem prompts. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*, SIGCSE '19, page 531–537, New York, NY, USA, 2019. Association for Computing Machinery.

[7] G. Polya. *How to solve it: A new aspect of mathematical method*, volume 85. Princeton university press, 2004.

[8] D. J. Barnes, S. Fincher, and S. Thompson. Introductory problem solving in computer science. In *5th Annual Conference on the Teaching of Computing*, pages 36–39, 1997.

[9] D. McCall and M. Kölling. Meaningful categorisation of novice programmer errors. In *2014 IEEE Frontiers in Education Conference (FIE) Proceedings*, pages 1–8. IEEE, 2014.

[10] D. McCall and M. Kölling. A new look at novice programmer errors. *ACM Transactions on Computing Education (TOCE)*, 19(4):1–30, 2019.

[11] M. Hristova, A. Misra, M. Rutter, and R. Mercuri. Identifying and correcting java programming errors for introductory computer science students. *ACM SIGCSE Bulletin*, 35(1):153–156, 2003.

[12] B. Eichmann, F. Goldhammer, S. Greiff, L. Pucite, and J. Naumann. The role of planning in complex problem solving. *Computers & Education*, 128:1–12, 2019.

[13] S. Brand-Gruwel, I. Wopereis, and Y. Vermetten. Information problem solving by experts and novices: Analysis of a complex cognitive skill. *Computers in Human Behavior*, 21(3):487–508, 2005.

[14] K. Falkner, R. Vivian, and N. J. G. Falkner. Identifying computer science self-regulated learning strategies. In *Proceedings of the 2014 conference on Innovation & technology in computer science education*, pages 291–296, 2014.

[15] C. M. Allwood. Novices on the computer: a review of the literature. *International Journal of Man-Machine Studies*, 25(6):633–658, 1986.

[16] D. Loksa, A. J. Ko, W. Jernigan, A. Oleson, C. J. Mendez, and M. M. Burnett. Programming, problem solving, and self-awareness: Effects of explicit guidance. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, page 1449–1461, New York, NY, USA, 2016. Association for Computing Machinery.

[17] L. E. Margulieux, R. Catrambone, and M. Guzdial. Employing subgoals in computer programming education. *Computer Science Education*, 26(1):44–67, 2016.

[18] T. VanDeGrift, T. Caruso, N. Hill, and B. Simon. Experience report: Getting novice programmers to think about improving their software development process. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, pages 493–498, 2011.

[19] S. H. Edwards, J. Snyder, M. A. Pérez-Quiñones, A. Allevato, D. Kim, and B. Tretola. Comparing effective and ineffective behaviors of student programmers. In *Proceedings of the Fifth International Workshop on Computing Education Research Workshop*, ICER '09, page 3–14, New York, NY, USA, 2009. Association for Computing Machinery.

[20] Y. Chun Tie, M. Birks, and K. Francis. Grounded theory research: A design framework for novice researchers. *SAGE open medicine*, 7:2050312118822927, 2019.

[21] V. Fix, S. Wiedenbeck, and J. Scholtz. Mental representations of programs by novices and experts. In *Proceedings of the INTERACT'93 and CHI'93 conference on Human factors in computing systems*, pages 74–79, 1993.

[22] E. Charters. The use of think-aloud methods in qualitative research an introduction to think-aloud methods. *Brock Education Journal*, 12(2), 2003.

[23] Q. Hao, D. H. Smith IV, N. Iriumi, M. Tsikerdekis, and A. J. Ko. A systematic investigation of replications in computing education research. *ACM Transactions on Computing Education (TOCE)*, 19(4):1–18, 2019.

Appendix

Survey

Question 1:
What year of university are you in?

- 1st year
- 2nd year
- 3rd year
- 4th year
- 5th year
- Graduate student

Question 2:
What gender do you identify as?

- Male
- Female
- Non-binary
- Not listed
- Prefer not to answer

Question 3:
What is your age?

- 18
- 19
- 20
- 21
- 22
- 23
- 24
- 25 or above
- Prefer not to answer

Question 4:
Please specify your ethnicity.

- Caucasian
- African-American
- Latino or Hispanic

- Asian
- Native American
- Native Hawaiian or Pacific Islander
- Two or more
- Other/Unknown
- Prefer not to answer

Question 5:
Before this quarter, when did you take your last programming course?

- This is my first programming course
- Last quarter
- 2 quarters ago
- 3 quarters ago
- 4 quarters ago
- 5 quarters ago
- 6 quarters ago
- More than 6 quarters ago

Question 6:
How many programming courses have you previously completed?

- 0
- 1
- 2
- 3
- 4
- 5
- 6 or more

Question 7:
If you answered 1 or more to the previous question, please list the prior programming courses you have taken here:

Question 8:
Please rate how much you agree with the following statement: Planning is an important part of programming.

- Strongly disagree
- Somewhat disagree
- Neither agree nor disagree

- Somewhat agree
- Strongly agree

Question 9:
Please rate how much you agree with the following statement: I am comfortable with planning before beginning a programming project.

- Strongly disagree
- Somewhat disagree
- Neither agree nor disagree
- Somewhat agree
- Strongly agree

Question 10:
Please rate how much you agree with the following statement: I am satisfied with my planning process for programming projects.

- Strongly disagree
- Somewhat disagree
- Neither agree nor disagree
- Somewhat agree
- Strongly agree

Question 11:
Please rate how much you agree with the following statement: I can produce code that correctly solves a problem in my computer science courses.

- Strongly disagree
- Somewhat disagree
- Neither agree nor disagree
- Somewhat agree
- Strongly agree

Question 12:
Please rate how much you agree with the following statement: I have significant prior programming experience.

- Strongly disagree
- Somewhat disagree
- Neither agree nor disagree
- Somewhat agree
- Strongly agree

Question 13:
Please rate how much you agree with the following statement: Overall, I consider myself a novice programmer.

- Strongly disagree
- Somewhat disagree
- Neither agree nor disagree
- Somewhat agree
- Strongly agree

Question 14:
Please rate how much you agree with the following statement: Overall, I consider myself an expert programmer.

- Strongly disagree
- Somewhat disagree
- Neither agree nor disagree
- Somewhat agree
- Strongly agree

Question 15:
Please describe your planning process used with as much detail as possible.

Question 16:
Reflecting on your planning process, is there anything that you would have done differently?

Question 17:
Please rate your level of detail for the following question: How detailed would you consider your planning for a typical programming assignment to be?

- Not very detailed
- Somewhat detailed
- Very detailed

Question 18:
Please describe how you decompose/break down the problem for a typical assignment with as much detail as possible.

Question 19:
Please rate the level of difficulty for the following question: How difficult do you find programming assignments on average?

- Extremely easy
- Somewhat easy

- Neither easy nor difficult
- Somewhat difficult
- Extremely difficult

Question 20:
Approximately how much time do you spend on an assignment?

Question 21:
Of the time spent, how much of that time would you estimate was spent on planning?

Question 22:
What grade do you typically receive on these assignments?

- A
- A-
- B+
- B
- B-
- C+
- C
- C-
- D+
- D
- F
- Prefer not to answer

Interview Questions

- Can you please tell me about your programming background? Please go into detail about how much programming experience you have and where that experience comes from.
- Can you please describe a recent programming assignment that you worked on for one of your courses?
- Please describe the process you used to prepare and plan for the programming assignment you just described? How much time did you spend on this?
- In your planning, what specific tools or techniques did you use, and how did you use those to better plan things out?
- Looking back on it, is there anything that you would have changed about your planning process for this programming assignment?
- Can you please describe in as much detail as possible, how you broke down/decomposed this specific problem?
- If you don't mind sharing, how well did you do or do you expect to do on this programming assignment? Did your expectation match how well you thought you did?

- How important do you think the planning process is to programming assignments? Please go into detail about why you think that as well.
- As you have gained more programming experience, do you think that how much and how you plan has changed? If so, can you describe those changes?
- How would you describe the level of detail you personally tend to use in the planning process for programming assignments? Please go into detail about why you think that as well.