Exploring the Impact of Exposing Command Line Programming to Early CS Majors (An HBCU Case Study)

Edward Dillon, Morgan State University

Dr. Dillon received his B.A. in Computer and Informational Science from the University of Mississippi in 2007. He would go on to obtain his Masters and Ph.D. in Computer Science from the University of Alabama in 2009 and 2012, respectively. Dr. Dillon is a newly tenured Associate Professor in the Department of Computer Science at Morgan State University. Prior to his arrival to Morgan State, Dr. Dillon served as a Computer Science Instructor at Jackson State University (2012-2013), and a Postdoctoral Researcher at Clemson University (2013-2014) and the University of Florida (2014-2016). His research focuses on human-centered computing, computer science education, social computing, and broadening participation in computing. Dr. Dillon has received >$750k in research funding and awards from external agencies and non-profit organizations, including the National Science Foundation (NSF), the Maryland Pre-Service Computer Science Teacher Education Program (MCCE), and the Collaborative Research Experience for Undergraduates (CREU - CRA-WP). Dr. Dillon currently serves as a Co-PI for the STARS Computing Corps, which recently has been renewed for funding by NSF. He has also conducted a Faculty in Residency at Google during the summer of 2018 to learn more about this company's culture, practices, and to understand the expectations for candidates (e.g. aspiring CS majors) who pursue career opportunities at this company and related prominent companies in tech.

Krystal L. Williams, University of Georgia
Ashley Simone Pryor, Morgan State University

College junior and Vice President of the Society for the Advancement of Computer Science, a Morgan State University ACM chapter. Active member of Morgan State's Women in Computer Science organization.

Theodore Wimberly Jr., Morgan State University
Mariah McMichael, Morgan State University
Abisola Mercy Arowolaju
Donald Bernard Davis, Morgan State University
Toluwanimi Ayodele, Morgan State University

Exploring the Impact of Exposing Command Line Programming to Early CS Majors (An HBCU Case Study)

Abstract

Learning to program is an essential part of developing computational skills amongst computer science (CS) majors. Yet, CS majors can encounter programming as a barrier and in many cases leave the field altogether. The learning process that CS majors encounter while developing their programming skills is multifaceted. They are expected to: 1) grasp necessary programming concepts, paradigms, and data structures, 2) become adept with employing the appropriate syntax and semantics for a given programming language used for code development, and 3) exhibit a proficiency for effectively operating a programming tool/editor assigned.

Effective approaches for teaching CS majors how to program has garnered much attention. When emphasizing the types of tools/editors that CS majors operate while programming in addition to these tools' potential impact to aid the students' development of their programming skills, much of this work seen in literature has focused on visual-based tools/editors that exhibit adequate affordance to the student for operation during code development (e.g. integrated development environment (IDEs)). In contrast, there are minimal studies that directly explore relative pedagogical impacts involving non-IDE based tools that are less visual in appearance and exhibit a lower operative affordance to the student (e.g. command-line based tools/editors). Moreover, there are even fewer studies that directly compare the impacts of IDEs and command line environments, respectively, on student learning.

This article discusses a case study that exposed early CS majors, enrolled in either CS2 or an Object-Oriented Programming course at a Mid-Atlantic Historically Black University in the United States, to command line programming. This study was conducted over a span of six semesters (Fall 2020 – Spring 2023). During these semesters, students in both courses, respectively, were assigned a command line workflow module to complete beginning in week 3. This module required these students to: 1) install a Unix-like command-line interface named Cygwin unto their personal computers, 2) learn a series of commands for navigating the file/directory system via the Cygwin interface, and 3) become proficient in building, compiling, and executing/interpreting programs while using the appropriate commands via the Cygwin interface. Students who owned a Mac-based computer were permitted to use the embedded Unix-based terminal interface on their Mac to complete this module.

To gauge the impact of command line usage on the students in these courses, two surveys were administered at different points of the semester to capture both performance and psycho-social related feedback about the students' experiences. Overall, it was revealed that the CS2 students tended to exhibit an unfavorable disposition towards command line programming, which was also reflected in their initial struggle to adjust to using a command line tool. On the other hand, the OOP students showed a better performance and disposition towards command line

programming, but this could have been influenced by acquired experiences both prior and external with using such tools.

1. Introduction

Developing ways to effectively teach early computer science (CS) majors how to program has been an important topic of interest for some time. When addressing student learning in early programming courses, there have been a variety of elements researched and observed, notable ones being: 1) the type of paradigms that are ideal for introducing students to programming [1], [2], [3], [4], 2) the simplicity or complexity of programming languages and how they may impact student learning [1], [5], [6], [7], and 3) the type of tools/editors that can be used to foster student learning and engagement [8], [9], [10].

This work contributes to the third element surrounding tools/editors and their potential impact on student learning as they are developing their computational programming skills. Current literature around this effort shows a substantial trend of highlighting visual-based environments like integrated development environments (IDEs) and their impact on student learning in early learning stages of programming. Yet, the existence of non-IDE tools, like command line environments, are also available for students to use while programming.

It has been argued that IDEs tend to provide a level of feature affordance, familiarity, and intuitiveness that makes them more feasible to use [11], [12]. However, there is still the concern of whether such environments may be guilty of feature scaffolding, which in turn gives students a misconception about the true functionality of computational programming, especially across multiple operating systems [13]. Chen and Marx note from their personal study that students may be able to acquire a better mental model towards programming by moving from an IDE to command line programming [14]. This raises this question regarding why more efforts have not been devoted to studying the overall potential impact of command line tools on early CS majors as they are learning to program.

Through a case study comprising six semesters at a Mid-Atlantic Historically Black University, this article contributes to exploring the impact of infusing command line programming into an introductory course (CS2) and an intermediate level object-oriented programming (OOP) course. Moreover, it provides a direct comparative study that explores the impacts of command line programming versus IDE and their respective impact on student learning. Details and outcomes pertaining to this case study are discussed after the Literature Review section.

2. Literature Review

2.1 Tools for Early CS Courses

Literature has provided much emphasis and empirical evidence as it pertains to programming tools and their impacts on early CS majors upon exposure. There has been a debate along with a series of studies addressing whether or not programming tools (or even computers altogether) should be introduced during the initial stages of computational learning and development [15], [16], [17]. Nevertheless, there have been a variety of initiatives for gauging the type of tools

appropriate to introduce in early programming courses. Kelleher and Pausch, for instance, developed a taxonomy for classifying computational programming tools on the basis of social learning, motivation/engagement, code understanding, language understanding, entertainment, and general education [10]. When placing a direct focus on pedagogical tools, or tools that are designed to foster better learning in introductory programming courses, visual tools like IDEs are found to be adopted for use more frequently [18], [19], [20], [21]. Command line tools are alternative applications that can be used as part of the learning process for early CS majors. Instructor preference, familiarity, or related factors could play a role in whether command line tools are adopted for use in introductory programming courses.

2.2 IDE vs. Command Line Programming

When comparing IDE programming to command line programming, one obvious difference are the feature sets used for code developing, code editing, tool operation, and tool navigation. Dillon et al. developed a continuum highlighting the feature sets that typically exist between IDEs and command line tools [11]. This continuum revealed that IDEs typically provide features that are more intuitive to the programmer and assistive during programming development (such as syntax highlighting, error highlighting, auto completion, and mouse usage). Moreover, the overall behavior and functionality of IDEs can be considered more familiar to users (including novice programmers) since they are typically constructed using a WIMP format (window, icon, menu, and pointing device) for interaction and operation. In contrast, command line tools typically consist of a terminal window that is operational through the use of embedded command sets. Even though there exist command line-based tools that encompass a subset of assistive features such as syntax highlighting and error highlighting [22], programmers are typically subjected to a restricted set of features for operation.

2.3 Tool Evolution: Web-Based IDEs & IDE-Command Line Hybrid Tools

Over the past decade, there has been the emergence of web-based IDEs to provide more of a convenience for programmers to develop computational solutions directly online [23], [24], [25]. Similar to traditional IDEs, these web-based IDEs can be language specific or can encompass a variety of programming languages for professional and pedagogical use. Collegiate instructors and researchers have even observed the value and overall potential of web-based IDEs for sustaining student learning during the COVID-19 pandemic [26] [27].

Similarly, there has been the emergence of IDE-command line hybrid tools. One possible reason for this tool hybrid is to combat the potential scaffolding that IDEs can offer students for convenience, but could also impede certain aspects of their mental model development for programming, such as debugging and a broader understanding of programming concepts [13], [14], [28]. In contrast, command line environments can show a deficiency for being intuitive towards novice users, restrict feature affordance, and lack the necessary accessibility to be feasible for all users [29]. Therefore, it is possible that the existence of IDE-command line hybrid tools could provide the best of both worlds to programmers regardless of their level of programming experience.

3. Methods

The assessments used to examine the students' performance and experience with command line programming were two-fold: 1) *a command line quiz* and 2) *a pedagogical coding review (PCR)* assessment. Both assessments are discussed in the following subsections. The targeted participants for these assessments were students enrolled in either the CS2 or Object-Oriented Programming (OOP) course at a Mid-Atlantic HBCU in the United States. Both assessments were conducted over a span of 6 semesters ranging from Fall 2020 to Spring 2023. *Just to note: the OOP course was not offered during the Fall 2021 semester, therefore the results and findings for this specific course only reflects five of the six semesters (Fall 2020, Spring 2021, Spring 2022, Fall 2022, and Spring 2023).* At this Mid-Atlantic HBCU, CS2 students are traditionally taught intermediate programming concepts and data structures using Python. The students enrolled in the course are primarily freshmen CS majors or non-CS majors who need a CS course as part of their graduation requirement. The OOP students are traditionally taught advanced programming and object-oriented structures using C++. The students enrolled in this course are primarily sophomore and junior CS majors. From a demographic standpoint, the students typically enrolled in both courses are predominantly Black/African American.

3.1 Command Line Workflow

The objective of the command line workflow was to examine the students' ability to adapt to using a command line tool to: 1) navigate and maneuver a file system without the familiar use of WIMP features (as seen with IDEs), and 2) perform computational programming tasks through the strict usage of command sets. The broader objective of this workflow was to expand the students' knowledge of computational navigation and programming development while expanding upon their technical skill sets. During week 2 of a given semester, students in both CS2 and OOP courses were introduced to the command line workflow module via Canvas. This module was comprised of two protocols that the students underwent during their exposure to command line programming: 1) file navigation practices via a Linux-based command line tool, and 2) a document comprised of two exercises that train them with composing, editing, compiling, and running program solutions via a Linux-based command line tool. Since this workflow was conducted outside of class, the students were required to use their personal laptops to complete the protocols. For students who owned a PC or a related Windows-based laptop, they were instructed to download Cygwin, which is a Windows-based command line system that uses Linux-based scripts and commands for usage [30]. For students who owned either a Mac or Linux-based laptop, they were required to download packages directly onto their command terminal for usage.

3.2 Command Line Quiz

During week 4 of a given semester, the students received a quiz composed of a series of procedures that they needed to conduct via Cygwin or Mac/Linux-based terminal. These procedures consisted of file navigation protocols and programming composition/debugging procedures in Python (CS2) or C++ (OOP), respectively. The students were given 45 minutes to complete this quiz. Afterwards, they would submit their solutions to Canvas. Finally, a survey was administered using SurveyMonkey to gather detailed feedback about their experiences with

command line usage. Through this survey there were eight factors examined to capture the students' experience with using a command line tool for programming: *prior experience, completion rate, initial response, easy attributes, hard attributes, comfort level, confidence level,* and *fondness level*. Outcomes from these factors are discussed in further detail in the Results section.

3.3 Coding Review Assessment

During week 8 of a given semester, the students underwent one of two pedagogical coding review (PCR) assessments that examined their computational thinking skills based on the material covered in the CS2 and OOP courses, respectively, at that point during the semester. Since the COVID-19 pandemic was at its peak during most of these semesters, these assessments were administered via Zoom. In both courses, students were divided into groups of two and assigned a breakout room to conduct the PCR assessment. In their assigned groups, students received an electronic document consisting of two tasks to complete. Task 1 consisted of a program solution written in Python (CS2) or C++ (OOP) for the students to review. The students were instructed to: 1) examine this solution for any syntactic and semantic errors *(code reliability and correctness)*, and 2) determine the intended output that this solution entailed *(code behavior/functionality)*. For Task 2, the students were given a problem to solve computationally from scratch. During both tasks, the students were required to discuss their approaches for solving them aloud. In each breakout room, the paired students were granted access to record their interaction for each task, and to share their personal computer/laptop screens to show their tool editor used. The time to complete both tasks were 30 minutes, respectively. Notable findings that strictly emphasized the PCR elements involved in these particular assessments have already been published [31].

As part of this PCR assessment, the students were assigned a specific programming tool to use, either Cygwin (or terminal for students with Mac/Linux-based laptops) or a visual/IDE tool (majority used the Repl.It web-based IDE) to complete task 1. For task 2, they were required to switch tools and use the other for completion. The intent of this approach was to expose all the students to command line programming at some point during this assessment. Just to note: for the second PCR assessment, which typically occurred towards the end of the semester, students were usually given the option to choose which programming tool to use. Therefore, the results discussed in the next section will only reflect the first PCR assessment.

4. Results

The results section provides descriptive data reflecting the performances of the students in CS2 and OOP, respectively, on the command line quiz and the first PCR assessment. These data reflect the students' responses to the administered surveys that accompanied both assessments, respectively. The objective was to capture how the students were performing during these critical junctures in each semester while learning and employing command line-based programming. The next subsections, respectively, details the students' performance on both assessments based on the gathered feedback from these surveys. Just to note, there were students who elected not to complete one or either survey. Therefore, the total number of student participants, or N, listed in

the following tables fluctuates while reflecting the actual number of students who participated in each aspect of these assessments.

4.1 Command Line Quiz

Table 1 provides descriptive statistics about whether the students in either CS2 or OOP received any prior exposure to command line tools usage before undergoing the command line workflow and quiz. The percentages revealed that this was more of the case for the OOP students. Moreover, a McNemar Test was conducted to determine whether the number of students enrolled in OOP, who had prior experiences with command line programming, was significantly higher than the students enrolled in CS2. This test revealed this case to be true (*chi-squared = 14.187, df = 1, p-value = 0.0001655*).

Table 1. Prior Experiences with Using Command Line Tools (Fall 2020 - Spring 2023)

First Time Ever Using with Command Line Tools			
	N	**YES**	**NO**
CS2	**108**	81%	19%
OOP	**90**	57%	43%

It was expected that many of the students would need more than two weeks to master procedures for file navigation and developing computational programs while using command line protocols. Likewise, it was anticipated that many of them would struggle to provide the correct answers based on the series of protocols required for this quiz. Therefore, when observing the students' completion rates on the command line quiz in both courses, this was measured based on whether the students: 1) were able complete or attempt the quiz, or 2) fail to submit to a solution Canvas. Table 2 details the completion rate of the Command Line Quiz by course. The results revealed that most of the students in both CS2 (53%) and OOP (68%), respectively, were able to complete or at least attempt this quiz and submit their solutions to Canvas. A McNemar Test was conducted to gauge whether the OOP students had a higher submission rate of the command line quiz than their CS2 counterparts. This test revealed this case to be true (*chi-squared = 6.0822, df = 1, p-value = 0.01365*).

Table 2. Command Line Quiz - Completion Rate (Fall 2020 - Spring 2023)

Completion Rate			
	N	**Completed-Attempted**	**No Submission of Quiz**
CS2	**158**	53%	47%
OOP	**146**	68%	32%

Since command line usage was relatively new for many of these students, especially the CS2 students, it was important to capture their initial response or perception to this experience. Table 3 provides descriptive data regarding how these students initially perceived the nature of using command line tools. This question was captured in the form of an open-ended response. The

results revealed that most of the students in CS2 (69%) and OOP (49%), respectively, gave an unfavorable response to this question. When observing how frequent the CS2 students provided an unfavorable response in comparison to their OOP counterparts, a two-tailed T-Test revealed that such unfavorable responses were significantly higher amongst the CS2 students ($p<0.01$).

Table 3. Command Line Quiz - Initial Response (Fall 2020 - Spring 2023)

Initial Response				
	N	***Favorable***	***Unfavorable***	***Other***
CS2	107	22%	69%	9%
OOP	89	42%	49%	9%

The students were also asked to provide further details regarding concrete factors that influenced their perceptions and overall experiences with using a command line tool. This question was asked in two parts to capture both the *easy* and *hard* attributes of using their command line tool. Figures 1-4 detail concrete attributes that the students deemed easy or hard about using the command line tool. It was found that most of the CS2 students noted *command usage* as the easiest attribute, while *nothing/not easy* and *file access* were determined to be the second and third most noted easy attribute (Figure 1). Amongst the OOP students, *file access* was noted as the easiest attribute, while *command usage* and *feature usage* were noted as second and third, respectively (Figure 2). In turn, most students in both CS2 and OOP, respectively, note that *feature use* was the hardest attribute during this experience while *file access* and *learning curve* were second and third most noted for the CS2 students (Figure 3), and *command use* and *file access* were second and third most noted for the OOP students (Figure 4).

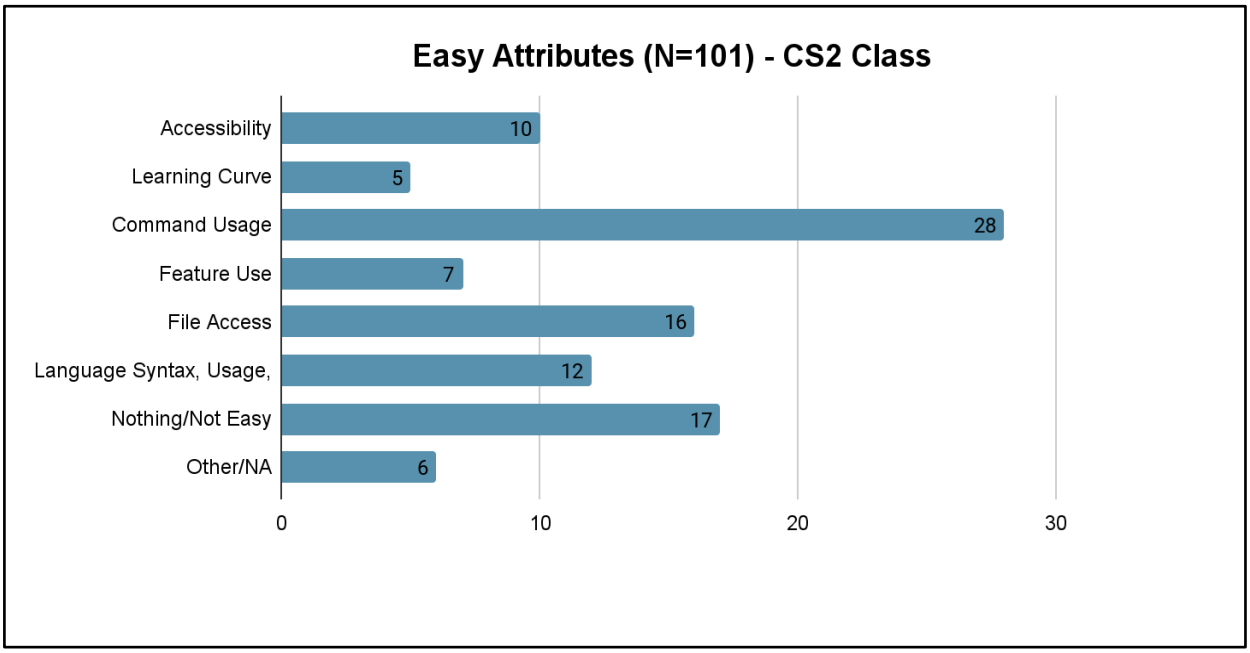

Figure 1: Easy Attributes of using a Command Line Tool - Descriptive Data (CS2)

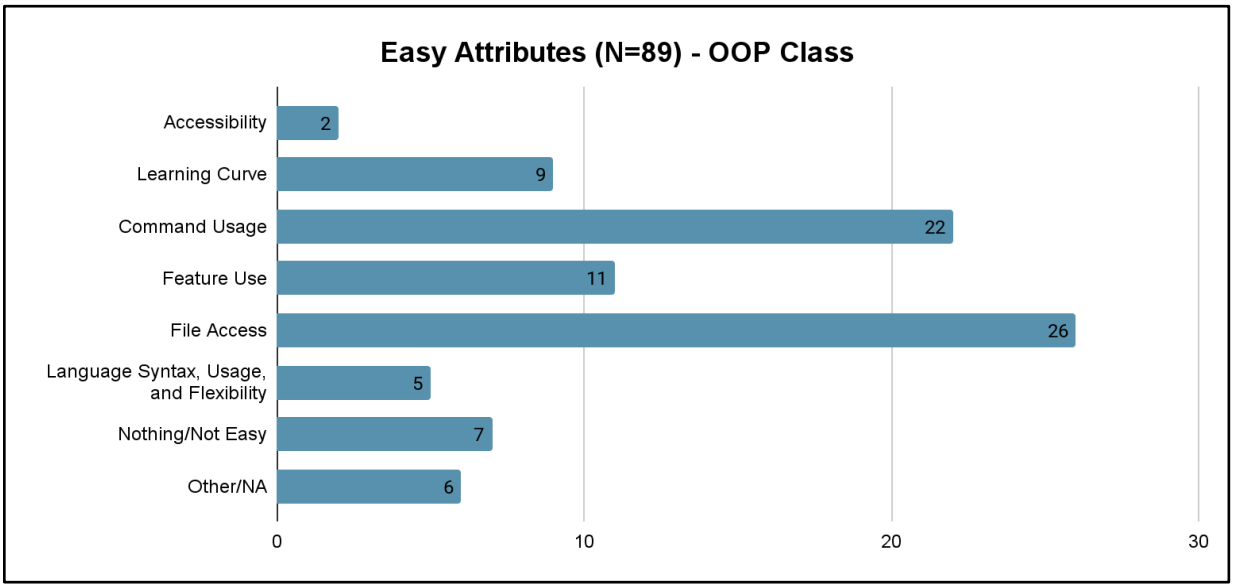

Figure 2: Easy Attributes of using a Command Line Tool - Descriptive Data (OOP)

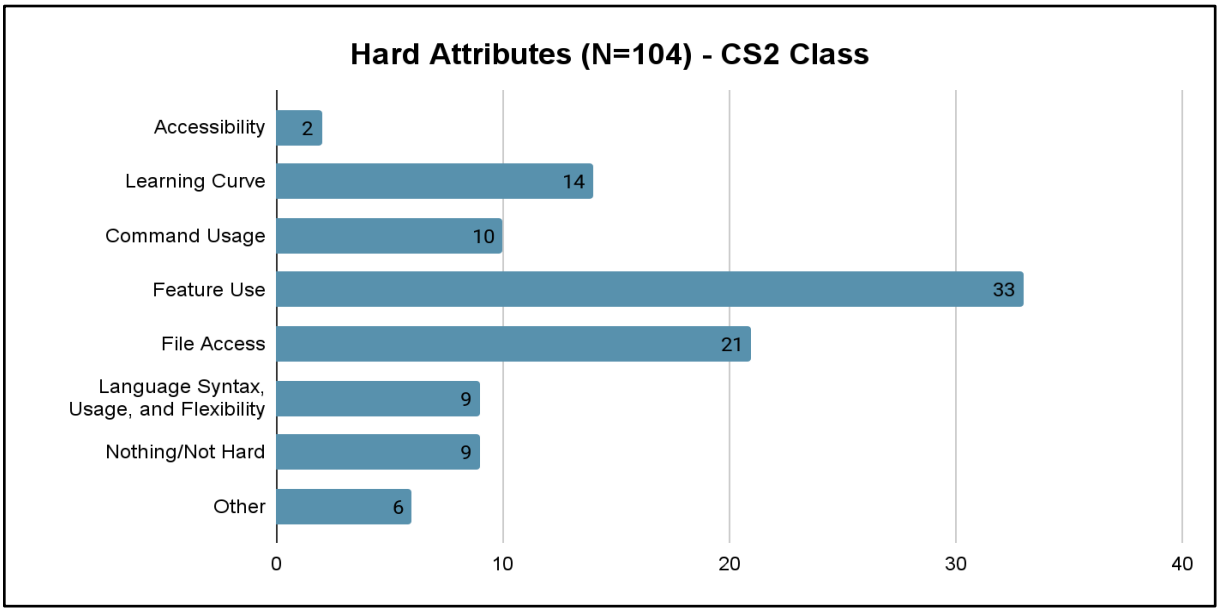

Figure 3: Hard Attributes of using a Command Line Tool - Descriptive Data (CS2)

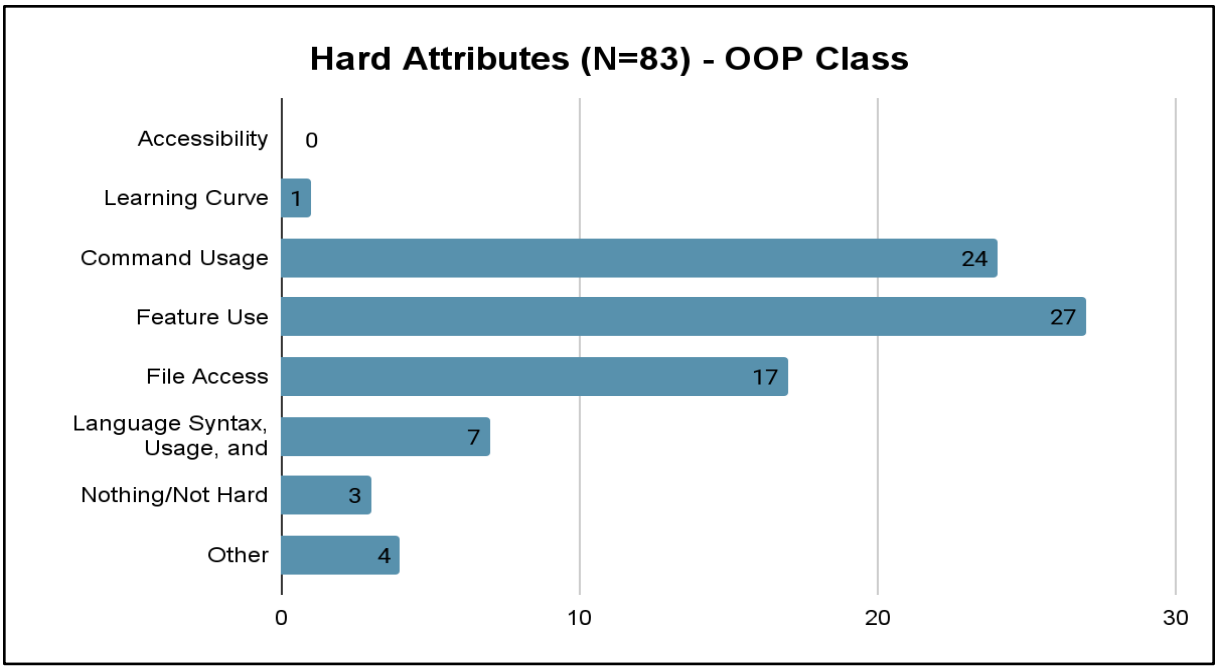

Figure 4: Hard Attributes of using a Command Line Tool - Descriptive Data (OOP)

There were also some psycho-social (or subjective) based questions asked of the students as it pertained to personal feelings toward their command line tool. This consisted of a series of questions that examined the students' comfort levels, confidence levels, and levels of fondness towards using a command line tool. These levels were based on a 7-point Likert scale. It was found in both courses that the students' level of confidence toward command line usage tended to be the lowest in most cases. The OOP students exhibited a higher level of comfort, confidence, and fondness toward command line tool usage than the CS2 students. A two-tailed t-test was conducted on all three psycho-social attributes to examine whether the OOP students' comfortability, confidence, and fondness towards command line programming were significantly higher than their CS2 counterparts. Each test revealed this to be true (*each test exhibited* $p<0.01$). Table 4 provides descriptive details regarding each of these three psycho-social factors.

Table 4. Comfort, Confidence, and Fondness Levels with Using Command Line Tool (Fall 2020 - Spring 2023)

Comfort Levels, Confidence Levels, and Fondness (7-Point Likert Scale)				
	N	***Comfort*** **Mean (with SD)**	***Confidence*** **Mean (with SD)**	***Fondness*** **Mean (with SD)**
CS2	107*	3.07 (1.91)	2.92 (1.95)	3.46 (1.86)
OOP	89	4.26 (1.75)	3.93 (1.83)	4.36 (1.99)
*** One student did not provide a response for *Comfort* and *Confidence*.**				

4.2 PCR Assessment

Table 5 provides descriptive details about the CS2 and OOP students' completion rates for each of the two tasks based on their assigned programming tool. Only 41% of the CS2 students, who were instructed to use a command line tool, completed Task 1. This percentage was much higher for the CS2 students using this tool to complete Task 2. For both Tasks 1 and 2 most of the OOP students, who were instructed to use a command line tool, were able to complete them.

Table 5. Assigned Tool/Editor(s) to Complete the Two Assigned Tasks
(Fall 2020 - Spring 2023)

Assigned Tool(s) - Command Line vs. IDE			
	N	Task 1	Task 2
CS2	77	Command Line: 41% IDE: 59%	Command Line: 71% IDE: 94%
OOP	85	Command Line: 71% IDE: 58%	Command Line: 60% IDE: 75%

Table 6 provides descriptive details regarding the students' initial response to their assigned tool for Task 1 portion of this coding review assessment. The results revealed that the majority of the CS2 students (63%), who were instructed to use a command line tool, exhibited an unfavorable response initially to its use, while the majority of their counterparts (70%), who were instructed to use an IDE, exhibited a favorable response. For the OOP students, a slight majority of them (40%) exhibited an unfavorable response to using a command line tool to complete Task 1, while the majority of their counterparts (86%) exhibited a favorable response to using an IDE.

Table 6. Initial Response to Assigned Tool (Command vs. IDE)
(Fall 2020 - Spring 2023)

Initial Response to Assigned Tool					
	N	Favorable	Unfavorable	Other	N/A
CS2	85	CL: 29% IDE: 70%	CL : 63% IDE: 10%	CL : 0% IDE: 16%	CL : 8% IDE: 3%
OOP	85	CL: 37% IDE: 86%	CL : 40% IDE: 2%	CL : 14% IDE: 6%	CL : 9% IDE: 6%
CL = Command Line					

Similar to the command line quiz, the students were also asked to provide further details regarding the dynamics of using their assigned command line tool or IDE during this experience. This question was asked in two parts to capture both the *easy* and *hard* attributes of using either tool. Figures 5-8 detail key factors for both attributes in relation to both courses, respectively. It was found that the majority of the students in CS2 and OOP, regardless of their assigned tool,

noted *feature usage* as the easiest attribute (Figures 5 and 6). For the hardest attributes, *feature usage* was also found to be the hardest attributes for those CS2 and OOP students, respectively, who used the command line while the majority of their IDE counterparts noted *nothing/not hard* for this response (Figures 7 and 8).

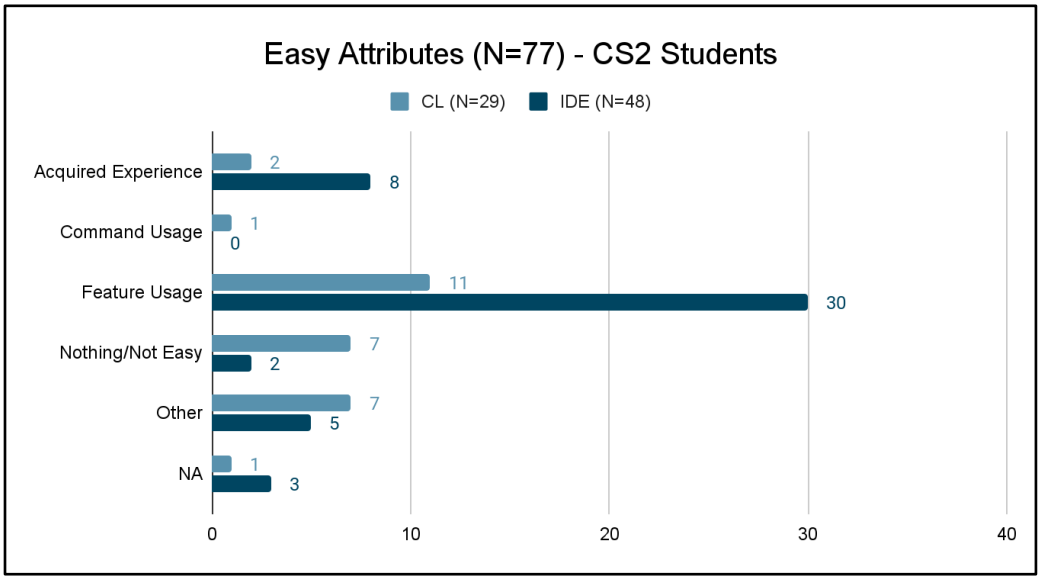

Figure 5: Easy Attributes of Using Assigned Tool (CS2) – based on Task #1

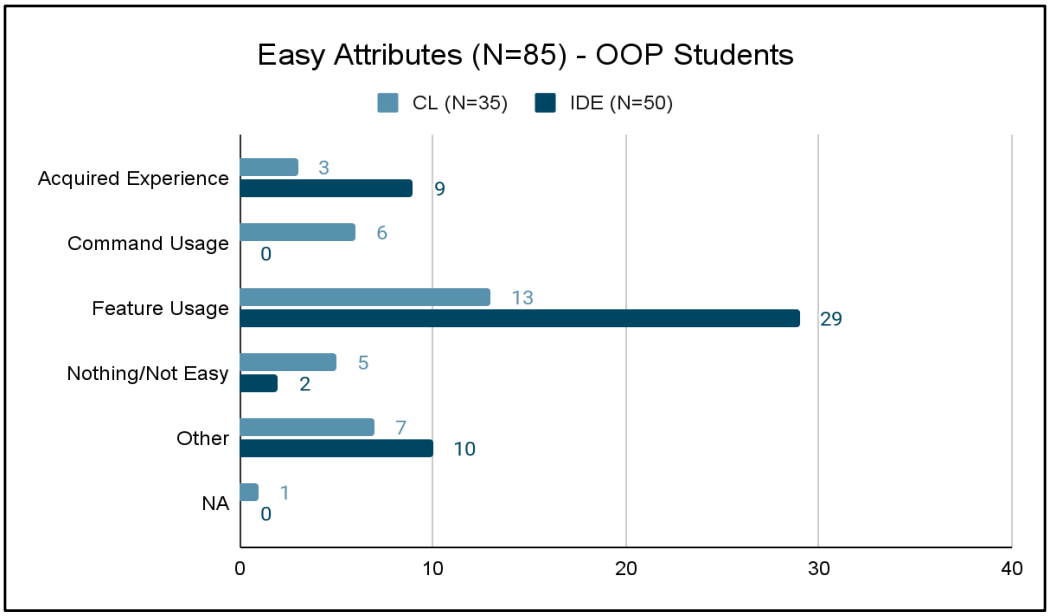

Figure 6: Easy Attributes of Using Assigned Tool (OOP) – based on Task #1

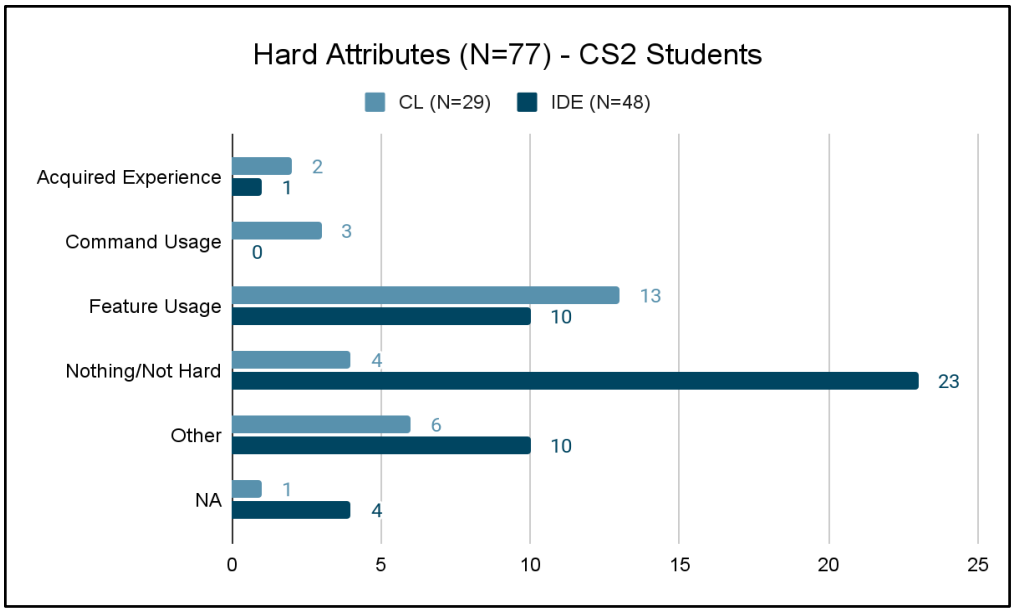

Figure 7: Hard Attributes of Using Assigned Tool (CS2) – based on Task #1

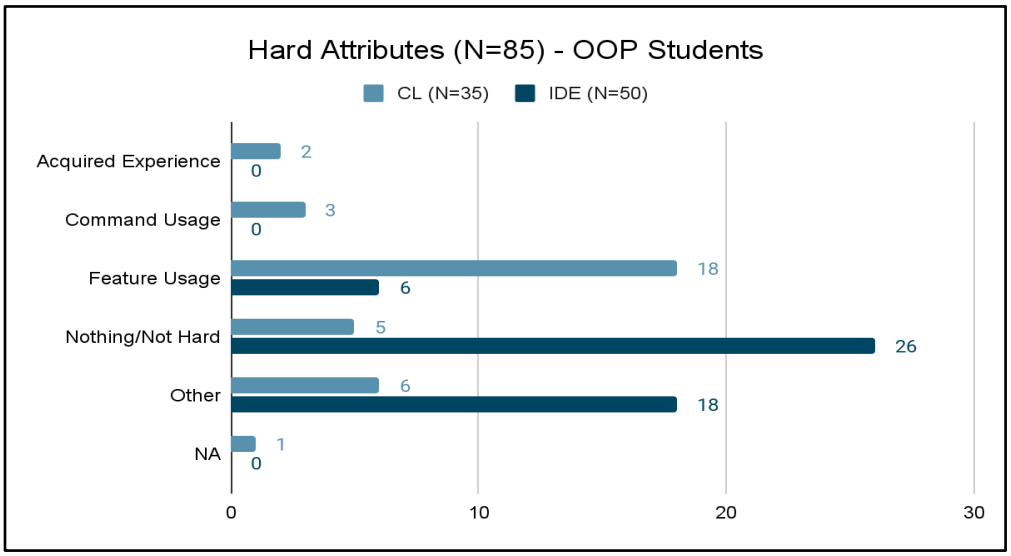

Figure 8: Hard Attributes of Using Assigned Tool (OOP) – based on Task #1

There were also a series of questions asked of the students to gauge their personal feelings towards using either tool. These questions examined the students' comfort levels, confidence levels, and levels of fondness towards using their assigned tool, respectively. These levels were based on a 7-point Likert scale. For the students who used a command line during Task 1, it was found that these students' fondness toward command line programming scored the lowest in both courses. In contrast, the fondness scores for students who used an IDE during Task 1 were found to be the highest of the three psycho-social attributes. Table 7 provides descriptive details regarding each of these three psycho-social factors.

Table 7. Comfort Levels, Confidence Levels, and Fondness with Using Assigned Tool (Fall 2020 - Spring 2023)

Comfort Levels, Confidence Levels, and Fondness with Using Assigned Tool (7-Point Likert Scale) – Mean Scores							
	N	Comfort		Confidence		Fondness	
CS2	77	Command Line (N=29)	Mean/SD 3.34/1.67	Command Line (N=29)	Mean/SD 3.07/1.67	Command Line (N=29)	Mean/SD 2.97/1.82
		IDE (N=48)	Mean/SD 5.60/1.40	IDE (N=48)	Mean/SD 4.98/1.55	IDE (N=48)	Mean/SD 5.81/1.53
OOP	85	Command Line (N=35)	Mean/SD 4.26/1.70	Command Line (N=35)	Mean/SD 3.91/2.09	Command Line (N=35)	Mean/SD 3.66/1.92
		IDE (N=50)	Mean/SD 6.02/0.94	IDE (N=50)	Mean/SD 5.36/1.41	IDE (N=50)	Mean/SD 6.16/1.02

5. Discussion

When observing these results in further detail, the prior experience that the OOP students collectively had with command line programming was evident. The majority of the OOP students were able to complete or at least attempt the command line quiz. Likewise, the majority of them were able to complete either task during the PCR assessment regardless of their assigned programming tool. In contrast, many of the CS2 students struggled with completing Task 1 when assigned a command line tool even though this rate improved drastically during Task 2. Moreover, a slight majority of them were able to complete or at least attempt the command line quiz, yet this percentage was significantly lower than their OOP counterparts' percentage.

Some of these disparities between the CS2 and OOP students were also seen when observing the psycho-social dynamics of these assessments, in particular during the command line quiz. It was found that the OOP students exhibited significantly higher levels of comfortability, confidence, and fondness toward using a command line tool than their CS2 counterparts. During the PCR assessment, the OOP students who used command line tools during Task 1 also showed a significantly higher level of comfortability and slightly higher levels of confidence and fondness than their CS2 counterparts. Finally, both cohorts of students showed higher rates of unfavorable perceptions towards using a command line tool during the command line quiz and PCR assessments, respectively. Likewise, the CS2 students exhibited significantly higher rates of unfavorable perceptions towards a command line tool than their OOP counterparts during the command line quiz.

When examining concrete attributes during these experiences that may have influenced these students' performance and perceptions during these case studies, it was found that some of these attributes were consistently noted as easy or hard by the majority of the CS2 and OOP students, respectively, despite their level of experience with command line tools. Case in point, the command line quiz revealed that *command usage* and *file access* were considered some of the

easiest attributes for both the CS2 and OOP students. At the same point, *feature use* was consistently found to be one of the hardest aspects of command line usage during both the command line quiz and the PCR assessment for both cohorts of students.

6. Threats to Validity

Even though the results and outcomes from this case study yield notable findings about the students' performance with command line programming in both courses, there are some potential threats to challenge the validity of these findings. One threat is the absence of interviews as part of this assessment. Majority of these results for this case study have been collected and analyzed from the surveys that accompanied both the command line quiz and coding review assessment, respectively. Another threat is the virtual setting used to administer these assessments. Some of these assessments were conducted during the heart of the COVID-19 pandemic which caused students to be quarantined. It is possible for these students to have used outside resources during the assessments that could not have been preventable. Another threat to this work is based on its scalability. Even though this work has the potential to be scalable to other institutions, the current outcomes from this case study only reflects one institution. Therefore, the results could be limited because of convenience sampling. When observing the level of experiences between the CS2 and OOP students, especially as it pertains to command line programming, it is possible that some of the OOP students acquired this experience as CS2 students in prior semesters of this case study.

Certain aspects of the PCR assessment also presented some potential threats that could challenge the validity of this work. One potential threat is the two-task structure of the PCR assessment. It is possible that the completion rates for Task #1 were lower due to the students' lack of experience with the coding review assessments. This could have also influenced an increase in their performance and completion rates in Task #2 after understanding how the coding review assessment works. Moreover, Task #2 itself also yielded another potential threat since the students' performance on this task was not examined in the same capacity as Task #1 even though the students were instructed to swap tools between tasks. Examining the students' performance in Task #2 could have provided more context (e.g. easiest and hardest attributes, comfort, confidence, and fondness) about the students' experience with using command line after initially using an IDE during Task #1 of the assessment and vice-versa. Some of the data points revealed during the PCR assessment also yielded a potential threat based on how many students actually used their assigned command line tool versus an IDE. There was a tendency for students to naturally use or even switch to using IDEs during the coding review assessment due to their frustration and struggle with navigating the command line tool which originally prevented them from completing the assigned task.

7. Conclusion & Future Work

The objective of this article was to disseminate notable findings as it pertains to the impact of command line usage amongst early CS majors. Over a span of these six semesters observed for this case study, these students collectively showed some potential to operate their command line tool even though many of them exhibited unfavorable perceptions towards it. The CS2 students specifically showed some initial struggle with adjusting to command line programming, but this

case study also revealed this cohort's potential to improve in their performance over time. This was the actual case for the OOP students. This study also highlights the importance of acquired experience with command line tools. Specifically, such experience could potentially improve the students' performance and possible dispositions toward command line programming over time as seen with the OOP students.

To expand upon this study, it is the plan to incorporate interviews at the end of the semester. This future work will allow the ability to further gauge the students' experience and perceptions toward command line programming upon completing either their CS2 or OOP course. Another future work would be to align the second PCR assessment with the first by assigning certain tools for the students to use rather than giving them the option to choose their preferred tool. This would allow an additional collection point at the end of the semester to further gauge the students' experience with command line tools. From an assessment standpoint, it is the plan to convert the virtual assessments to in-person assessments to control for any confounding factors related to outside resources these students could lean upon to assist them during these case study interactions. From a scalability perspective, it is also the plan to expand this work and seek collaborations with colleagues at other institutions who value the importance of this work, especially those with interests in programming pedagogy, computational learning, and technical skill development in the classroom.

Acknowledgements

This current work is funded by the National Science Foundation NSF-EDU 2011793.

References

[1] R. Brown, J. Davis, S. A. Rebelsky, and B. Harvey, "Whither scheme?: 21 st century approaches to scheme in CS1," in *Proceedings of the 40th ACM technical symposium on Computer science education*, Chattanooga TN USA: ACM, Mar. 2009, pp. 551–552. doi: 10.1145/1508865.1509055.

[2] O. Astrachan, K. Bruce, E. Koffman, M. Kölling, and S. Reges, "Resolved: objects early has failed," *ACM SIGCSE Bull.*, vol. 37, no. 1, pp. 451–452, Feb. 2005, doi: 10.1145/1047124.1047359.

[3] F. Bailie, M. Courtney, K. Murray, R. Schiaffino, and S. Tuohy, "Objects first - does it work?," *Consort. Comput. Sci. Coll.*, no. December 2003, pp. 303–305, 2003.

[4] S. Cooper, W. Dann, and R. Pausch, "Teaching objects-first in introductory computer science," in *Proceedings of the 34th SIGCSE technical symposium on Computer science education*, Reno Navada USA: ACM, Jan. 2003, pp. 191–195. doi: 10.1145/611892.611966.

[5] M. H. Goldwasser and D. Letscher, "Teaching an object-oriented CS1 -: with Python," in *Proceedings of the 13th annual conference on Innovation and technology in computer science education*, Madrid Spain: ACM, Jun. 2008, pp. 42–46. doi: 10.1145/1384271.1384285.

[6] Z. Dodds, C. Alvarado, G. Kuenning, and R. Libeskind-Hadas, "Breadth-first CS 1 for scientists," in *Proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education*, Dundee Scotland: ACM, Jun. 2007, pp. 23–27.

doi: 10.1145/1268784.1268794.
[7] A. Pears *et al.*, “A survey of literature on the teaching of introductory programming,” in *Working group reports on ITiCSE on Innovation and technology in computer science education*, Dundee Scotland: ACM, Dec. 2007, pp. 204–223. doi: 10.1145/1345443.1345441.
[8] C. M. Lewis, “How programming environment shapes perception, learning and goals: logo vs. scratch,” in *Proceedings of the 41st ACM technical symposium on Computer science education*, Milwaukee Wisconsin USA: ACM, Mar. 2010, pp. 346–350. doi: 10.1145/1734263.1734383.
[9] C. Kelleher, R. Pausch, and S. Kiesler, “Storytelling alice motivates middle school girls to learn computer programming,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, San Jose California USA: ACM, Apr. 2007, pp. 1455–1464. doi: 10.1145/1240624.1240844.
[10] C. Kelleher and R. Pausch, “Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers,” *ACM Comput. Surv.*, vol. 37, no. 2, pp. 83–137, Jun. 2005, doi: 10.1145/1089733.1089734.
[11] E. Dillon, M. Anderson, and M. Brown. “Comparing feature assistance between programming environments and their "effect" on novice programmers.” *J. Comput. Sci. Coll.* 27, 5, May 2012, pp. 69–77.
[12] E. Dillon, M. Anderson, and M. Brown. “Comparing mental models of novice programmers when using visual and command line environments,” in *Proceedings of the 50th Annual Southeast Regional Conference (ACM-SE '12)*. *Association for Computing Machinery,* New York, NY, USA, pp. 142–147, March 2012, doi: 10.1145/2184512.2184546.
[13] E. Dillon, M. Anderson-Herzog, and M. Brown. “Teaching students to program using visual environments: Impetus for a faulty mental model?.” *Journal of Computational Science Education*, 5,1 August 2014, pp. 28-43.
[14] Z. Chen and D. Marx, “Experiences with Eclipse IDE in programming courses,” *Consort. Comput. Sci. Coll.*, vol. 21, no. December 2005, pp. 104–112.
[15] F. Hermans and E. Aivaloglou, “To Scratch or not to Scratch?: A controlled experiment comparing plugged first and unplugged first programming lessons,” in *Proceedings of the 12th Workshop on Primary and Secondary Computing Education*, Nijmegen Netherlands: ACM, Nov. 2017, pp. 49–56. doi: 10.1145/3137065.3137072.
[16] V. A. Cicirello, “A CS unplugged activity for the online classroom,” *Consort. Comput. Sci. Coll.*, vol. 28, no. 6, pp. 162–168, Jun. 2013.
[17] T. Nishida, S. Kanemune, Y. Idosaka, M. Namiki, T. Bell, and Y. Kuno, “A CS unplugged design pattern,” in *Proceedings of the 40th ACM technical symposium on Computer science education*, Chattanooga TN USA: ACM, Mar. 2009, pp. 231–235. doi: 10.1145/1508865.1508951.
[18] “Alice.” Accessed: Feb. 07, 2024. [Online]. Available: http://www.alice.org
[19] “BlueJ.” Accessed: Feb. 07, 2024. [Online]. Available: https://www.bluej.org
[20] “Dr. Java.” Accessed: Feb. 07, 2024. [Online]. Available: https://drjava.sourceforge.net
[21] “jGrasp.” Accessed: Feb. 07, 2024. [Online]. Available: https://www.jgrasp.org
[22] “GNU Emacs.” Accessed: Feb. 07, 2024. [Online]. Available: https://www.gnu.org/software/emacs/
[23] “Replit.” Accessed: Feb. 07, 2024. [Online]. Available: https://replit.com/
[24] “W3Schools.” Accessed: Feb. 07, 2024. [Online]. Available:

https://www.w3schools.com/tryit/
[25] “Python Tutor.” Accessed: Feb. 07, 2024. [Online]. Available: https://pythontutor.com
[26] L. Sinanaj, J. Ajdari, M. Hamiti, and X. Zenuni, “A comparison between online compilers: A Case Study,” in *2022 11th Mediterranean Conference on Embedded Computing (MECO)*, Budva, Montenegro: IEEE, Jun. 2022, pp. 1–6. doi: 10.1109/MECO55406.2022.9797096.
[27] M. M. Rahman and R. P. Morgan, “A Remote Instructional Approach with Interactive and Collaborative Learning to Teach an Introductory Programming Course during COVID-19 Pandemic,” in *2021 International Conference on Computational Science and Computational Intelligence (CSCI)*, Las Vegas, NV, USA: IEEE, Dec. 2021, pp. 940–946. doi: 10.1109/CSCI54926.2021.00210.
[28] H. Aljafer and G. Cantrell, “Learning and Teaching Undergraduate Introductory Programming Courses in Java – The Use of an IDE VS Command Line,” in *2020 International Conference on Computational Science and Computational Intelligence (CSCI)*, Las Vegas, NV, USA: IEEE, Dec. 2020, pp. 992–997. doi: 10.1109/CSCI51800.2020.00184.
[29] H. Sampath, A. Merrick, and A. Macvean, “Accessibility of Command Line Interfaces,” in *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, Yokohama Japan: ACM, May 2021, pp. 1–10. doi: 10.1145/3411764.3445544.
[30] “Cygwin.” Accessed: Feb. 07, 2024. [Online]. Available: https://cygwin.com
[31] E. Dillon, B. Williams, A. Ajayi, Z. Bright, Q. Kimble-Brown, C. Rogers, M. Lewis, J. Esema, B. Clinkscale, and K.L. Williams. “Evaluating Face-to-Face vs. Virtual Pedagogical Coding Review Sessions in the CS classroom: An HBCU Case Study.” *2021 Conference on Research in Equitable and Sustained Participation in Engineering, Computing, and Technology (RESPECT)*, Philadelphia, PA, USA, 2021, pp. 1-5, doi: 10.1109/RESPECT51740.2021.9620586.