Analysis of Novices' Web-Based Help-Seeking Behavior While Programming

James Skripchuk*
North Carolina State University
Raleigh, North Carolina, USA
jmskripc@ncsu.edu

Neil Bennett†
North Carolina State University
Raleigh, North Carolina, USA
nzbennet@ncsu.edu

Jeffrey Zheng†
University of Pittsburgh
Pittsburgh, Pennsylvania, USA
jez43@pitt.edu

Eric Li
North Carolina State University
Raleigh, North Carolina, USA
eli7@ncsu.edu

Thomas Price
North Carolina State University
Raleigh, North Carolina, USA
twprice@ncsu.edu

ABSTRACT

Web-based help-seeking – finding and utilizing websites to solve a problem – is a critical skill during programming in both professional and academic settings. However, little work has explored how students, especially novices, engage in web-based help-seeking during programming, or what strategies they use and barriers they face. This study begins to investigate these questions through analysis of students' web-search behaviors during programming. We collected think-aloud, screen recording, and log data as students completed a challenging programming task. Students were encouraged to use the web for help when needed, as if in an internship. We then qualitatively analyzed the data to address three research questions: 1) What events motivate students to use web search? 2) What strategies do students employ to search for, select, and learn from web pages? 3) What barriers do students face in web search, and when do they arise? Our results suggest that that novices use a variety of web-search strategies – some quite unexpected – with varying degrees of success, suggesting that web search can be a challenging skill for novice programmers. We discuss how these results inform future research and pedagogy focused on how to support students in effective web search.

CCS CONCEPTS

• **Social and professional topics** → **Computing education**; • **Information systems** → *Search interfaces*; • **Software and its engineering**;

KEYWORDS

CS education, help-seeking, web-search, novice programming

*This material is based upon work supported by the National Science Foundation Graduate Research Fellowship under Grant No. DGE-2137100.
†Both authors contributed equally to this work and were supported by the National Science Foundation (Award No. 1950607).

SIGCSE 2023, March 15–18, 2023, Toronto, ON, Canada.

ACM ISBN 978-1-4503-9431-4/23/03…$15.00
https://doi.org/10.1145/3545945.3569852

ACM Reference Format:
James Skripchuk, Neil Bennett, Jeffrey Zheng, Eric Li, and Thomas Price. 2023. Analysis of Novices' Web-Based Help-Seeking Behavior While Programming. In *Proceedings of the 54th ACM Technical Symposium on Computing Science Education V. 1 (SIGCSE 2023), March 15–18, 2023, Toronto, ON, Canada.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3545945.3569852

1 INTRODUCTION

Programmers, both professionals and hobbyists, must constantly stay up to date with new languages, frameworks, and paradigms [15]. Programmers in industry and academia are expected to resolve their problems and learn concepts on their own in order to stay relevant. Students in upper level classes (such as Software Engineering) may be expected to resolve errors and self-learn "trivial" constructs of a new language (such as syntax and control structures) so that the class itself can focus on higher-level concepts. The World Wide Web serves as the most popular repository of knowledge for programming help, ranging from official documentation to question-answer websites [26].

However, despite the need for self-learning in upper level classes and post-graduation, there is little research in how students in programming related degree programs use the web for help, or how often web-based help-seeking skills are explicitly taught. Novices encounter frequent challenges in out-of-class activities, which they often attempt to address by searching online to learn new concepts or to debug their code [18]. The ability to effectively seek and integrate help - including web-based help-seeking - is a complex metacognitive skill that must be explicitly learned and practiced [10, 19, 20]. It is unclear how effective students' existing web-search strategies are, and students may be engaging in unproductive or even harmful help-seeking behaviors, as they do with other forms of help [16]. Understanding *how*, *why*, and *when* novices use the web would inform research on whether and how web-search should be explicitly taught, and how to support more effective web-search through tools and pedagogy.

Prior work on web-based help-seeking has focused almost exclusively on a software engineering context. These studies focus on how professional programmers (or upper-level students, serving as proxies for professionals) can solve problems more efficiently, rather than how students – especially novices – learn with web search. This leaves open questions about how and when web search

skills develop in students. A majority of web-based help-seeking literature, both in software engineering and in computing education, focus on coarse-grained surveys and quantitative analysis of data such as query logs. While useful, this type of analysis does not allow for robust and detailed insights into students' web-search behaviors or their perceptions of when web-search is appropriate [11].

By contrast, in this work, we qualitatively investigate how *novices* (in a second-level course) use web-based help-seeking to solve programming challenges. Undergraduate students ($n = 19$) completed a challenging programming task and were encouraged to use the web for help when needed, as if in an internship. Our aim was to study the role of web-based help-seeking across the entire programming process from start to finish, not just during debugging. We qualitatively analyzed think-aloud, screen recording, and log data using thematic analysis [5] in order to understand when, why and how students searched.

Our results suggest that that novices use a variety of web-search strategies with varying degrees of success, suggesting that web search can be a challenging skill for novice programmers. We discuss how these results inform future research and pedagogy. Our findings address the following research questions:

RQ1: What events motivate students to use web search, and when do they avoid it?

RQ2: What strategies do students employ to search for, select, and learn from web pages?

RQ3: What barriers do students face, and when in the web search process do they arise?

2 RELATED WORK

2.1 Theoretical Foundation: Help-Seeking

Help-seeking - the process in which learners recognize, identify, seek out, implement, and evaluate help - is a critical and complex metacognitive skill across all learning domains. Nelson-Le Gall discusses the importance of help-seeking as an instrumental skill which promotes the dissemination of knowledge in the classroom as well as building strong problem-solving strategies in cognitive and academic tasks. Nelson-Le Gall then provides a theoretical model of the process of help-seeking, which we use as a guide for our analysis [19]. The basic model proposed consists of a student's 1) Awareness of need for help, 2) Decision to seek help, 3) Identification of potential helper(s), 4) Employment of strategies to elicit help, and 5) Reactions to help-seeking attempt(s). Newman further expanded on this by introducing concepts of Self-Regulated Learning (SRL), and how the decision to seek help is related to the student's self-efficacy [20]. Traditional help-seeking models were designed within a classroom context, which have their own challenges, such as students being afraid or embarrassed to let others know they need help [13, 22]. We expect to see much of the same phases in the stated models during the web-based help-seeking process - except in this case the "helper" is replaced by the websites

These prior help-seeking models are useful, but general. In this work, we build on and expand prior work, focusing on a context where students can seek help without having to interact with another person (e.g., with the web or automated tutors). While studying help-seeking in automated tutors, Price et al. concluded that help-seeking might not just be in reaction to student difficulty, but also a valid problem solving strategy [21]. Work is needed to elaborate on the specific help-seeking strategies and barriers that arise in web search, which likely differs from classroom-specific sources of help, like instructors, TAs, and peers.

2.2 Professional Search Behavior

The focus on web-search in software engineering and HCI research has focused on topics such as the frequency of their web search [17], the content of their searches [26], and how they use code they find [2, 25]. Brandt et al. note that programmers engage in just-in-time learning of new skills during programming [4]. Programmers will often use the web as a substitute for memorization, use it to clarify implementation details, and even learn entirely new concepts. Hora details what professional software developers are searching for and what they find. The study finds characteristic traits of the queries used by professionals, such as using short queries, beginning queries with keywords such as the language name, and omitting functional words [12]. However, there is little research on how programmers learn and cultivate these web-based help-seeking skills and behaviors. Studying the web-based help-seeking behaviors of novice programmers can help us understand *how* and *when* these skills develop. While this research provides insight on the help-seeking behaviors of experienced and professional programmers, we aim to investigate if novices also engage in the same methodologies or struggles.

2.3 Novice Search Behavior

There is little prior work on how novices use the web for help-seeking. Li et al. investigated the differences between novices' and experts' web-based debugging behaviors, and they found that professional developers are more likely to find relevant results than novices [14]. Chatterjee et al. instructed novice software engineers to highlight parts of online examples they found relevant, and noted that they tended to highlight code examples more often than text [6]. However, these two studies limited themselves to StackOverflow.com and not the entirety of the web, and even though it is a popular source among students, Doebling et al. found that they had trouble navigating and filtering the sheer amount of information on the site [7]. Muller et al. studied novices' help-seeking behavior on homework, and noticed that students made use of the web in addition to official resources such as class resources and Piazza, and may be an integral portion of problem-solving process [18]. Al-Sammarraie showed that novice-expert pairs lead to novices constructing more helpful search queries than they would alone [1]. While helpful, very little of this prior work investigates the *behaviors* of novices throughout their entire programming process, not just debugging. We wish to study what behaviors students engage in outside of a traditional classroom or homework setting, where they do not have access to previous resources.

3 METHODOLOGY

Our goal in this work was to study the behaviors that novices engage in while using web-based help-seeking. In order to achieve this, we performed a think-aloud study where students solved a

programming task that was challenging but appropriate for their skill level.

3.1 Population & Course Context

We recruited students for this IRB-approved study from within a R1 university on the east coast of the U.S. We restricted our population to students who were currently enrolled in (but had not completed) the university's 200-level (CS2) course. The CS2 course is a continuation of the CS1 course, and both courses use Java as their programming language. CS2 students were chosen because they were still early in their CS program (i.e. novices), but were also familiar with the fundamentals of programming and were beginning to learn more complex programming patterns and APIs, which might benefit from online search for documentation and examples. The CS2 course covered Object-oriented programming, Input/Output (I/O), introduction to the software development lifecycle, recursion, and linear data structures. This course, as with a majority of the university's other CS courses, had restrictions on usage of external sources such as the web. For homeworks and projects, the only external resources students were allowed to use was the official Java API documentation. The use of any other external resources and help websites (even if cited) would be counted as an academic integrity violation if discovered. We discuss the implications of this in Section 5.

At the beginning of the semester, recruitment materials were sent out to the CS2 course instructor in addition to a researcher visiting the class and explaining the study. For completion of the study, subjects were given a $30 Amazon gift card. In total, 19 undergraduates signed up for and completed the study. These included CS majors (13/19), CS-intended (not yet accepted into the program; 3/19) and other STEM majors currently taking the class for the CS minor (3/19). They noted that their first programming experience was in: Middle School (2/19), High School (8/19), or Undergraduate (8/19)[1]. We did not collect other demographic information.

3.2 Procedure

Each study session was performed remotely over Zoom. Participants were allotted 60 minutes for the entire study. First, the researcher first spent 5 minutes overviewing the study consent and data collection procedures. Then, subjects were given approximately 10 minutes for an the interview portion on their usage and perceptions of web-search, and approximately 40 minutes for the programming portion. Whether or not the student finished the programming portion, the last 5 minutes was allotted to the researcher asking questions about behaviors that the researcher noticed during the programming portion. For the programming portion, participants were required to use a researcher-hosted VS Code server which logged programming events. They were also required to download a custom Chrome web-extension in order to track their search behaviors for the duration of the session. All 19 studies were conducted by a single researcher. Due to space constraints, in this paper we focus only on the programming portion of our study.

[1]We did not manage to get first exposure to programming for one student.

3.3 Programming Think-Aloud Study

We chose the Rainfall Problem as the basis for our study due to its precedent in the literature and how it requires core programming concepts [23]. In order to make the problem more difficult and to encourage search, the problem was modified to require programming APIs and skills (some of which students had recently learned in class), specifically: reading and writing CSV files, rounding, and using Dates. The programming task was scaffolded into 5 distinct steps, each indicated by comments in an otherwise empty Java file: 1) Extract the rainfall data from a column in a CSV, 2) Calculate the "Rainfall Average", 3) Round the Rainfall Average to two decimal places, 4) Get the current date, 5) Append both the current date and the formatted Rainfall Average to a different CSV file.

File I/O was added not only because it was a historically difficult topic that was covered in the CS2 class, but also because the web provides different approaches to read a file in Java, which can highlight how students choose a solution when multiple approaches are given. Rounding and getting formatted dates are not explicitly covered in the CS2 course, so this step would likely require searching online for some sort of help or documentation.

The programming context was framed as if it were an internship. Students were prompted that they were *"working for a meteorologist analyzing rainfall patterns in their area."* To discourage students from struggling with the problem by themselves and to encourage help-seeking, we stated that the *"quarterly report deadline is coming up, so your boss wants this program completed as soon as possible."* Students were explicitly told that they were *"allowed to use the Internet and any resources you find on the Internet to help you solve your problem."* and that *"you don't have to wait until you're stuck; you can look for help even before you start programming."* These comments were made in an effort to remind the student that they should not think of this as a class setting where external resources are discouraged. As part of the standard think-aloud study protocol, students were instructed to *"continuously talk about your thought process and what you're doing"*, and that if they were not speaking for a long amount of time (1 minute), they would be prompted by the researcher to begin thinking-aloud more. In order avoid periods of unproductive non help-seeking behavior, the researcher would occasionally prompt students in addition to the standard think-aloud prompting. If students explicitly stated they were stuck and did not know how to continue, or were engaging in unproductive behaviors without using the web (such as not thinking-aloud while making minimal edits) for more than 5 minutes, they were prompted by the researcher that *"as a reminder, you can use the web for help."*

3.4 Analysis

For this paper, we only focus on the programming portion and not the interview portion. After we collected and cleaned the data, we then proceeded with our analysis. We used the thematic analysis framework [5] to identify students' help-seeking triggers, strategies and barriers, similar to prior computing education work [8, 24].

First, three researchers (including the one who facilitated the interviews), familiarized themselves with the data by watching replays of 2 studies together, discussing what behaviors they were seeing, and writing down high-level notes on behaviors they saw.

The second step was to generate our initial codebook of behaviors. Thematic analysis requires fixed and unambiguous "units" to apply codes on. For interviews and transcripts, the units are usually utterances or line-by-line; however, since we were focused on recordings of programming behaviors and web-search, we defined our "units" with respect to these behaviors. Therefore, we used our log data to unambiguously segment each replay into chunks of time, and created a new segment each time a student compiled, made a web-search, or visited a page. In each segment of time, a researcher could then apply one or more codes detailing the behaviors they noticed within that segment. Due to our logging structure, creating segmentations in this manner is consistent, reproducible, and easily automated. Using their notes, the three researchers then performed open coding on the previous 2 replays and 3 new replays, ideating and apply codes to the segmentation, creating a final codebook with 34 codes in total.

Third, the researchers then grouped these codes into 8 distinct themes based on when and where these codes appeared within the data. After this step, two researchers then individually coded the remaining 14 replays. After each replay, the two researchers then met together and resolved differences and disputes on whether a code was present in a specific segment or not, before deciding upon a final coded segmentation for each replay in the closed coding process. The researchers then reviewed their themes, and made alignments with traditional help-seeking models, before defining them in detail. Across 19 students, 661 individual segments were coded, some segments having 0 codes while the maximum having 7 codes. In total, 663 codes were applied, with a median of 1 code per segment.

4 RESULTS & DISCUSSION

We now present some of our themes and codes. For space concerns, we do not discuss all of our codes and highlight codes (shown in bold) throughout this section. For each code, we report the number of students where the code appeared at least once. The codes and themes discussed here are chosen due their high frequency within our dataset (i.e. above the median frequency), with lower frequency codes included for contrast or to elaborate on unexpected behaviors. We provide a link to our resources containing all of the codes we encountered, as well as their frequency within the dataset[2]. Provided quotes may be edited for brevity, keeping all content but removing false starts and filler words. We detail our results and providing implications for both pedagogy and further research.

4.1 RQ1: Events That Lead To Search

4.1.1 First Search. We wanted to know what events motivated students to use web search, particularly, when and why they decide to make their first search during the programming process. We first look at when students made their first search. At the start of the programming portion, 6 students immediately searched for information on file I/O before writing a single line of code. In 3 of these pre-programming searches, students stated they had an approach already in mind but needed to look up the syntax for how to program them, using queries such as "`fileinputstream java`", and "`scanner java documentation`". P7, after rereading the problem statement, immediately opened up their search engine and typed in "`java scanner input`" while saying *"Right now I'm looking up Scanner to refresh myself on it."* The other 3 queried a more general problem statement, such as "`java read file`", and "`how to read a csv file java`". P9, after reading the prompt, stated that *"I'm not great at remembering how you can read data."* and began to type "`how to read data from a csv file in java.`".

The remaining 13 students started writing without any searches beforehand, while using a given approach from memory (such as `Scanner`, `BufferedReader`, or `FileInputStream`). After programming, but before making any compilations, 6 of these 13 students reached a point where they made a search, all refreshing themselves on the approach they were currently working towards. P17 was writing code to initialize an array, paused mid-line, and cautiously said *"I'm thinking this is how I do this? I'm trying to remember how to write a list of integers."*, before opening their browser and searching to confirm their code was correct. After programming, the remaining 7 of these 13 students did their first search after running their program encountering some sort of error (either compiler or logical error). Some would search for generic documentation pages (such as "`scanner api`") to help them, while others with a defined compiler error would use the error text to to help resolve their issues.

Discussion: We see that 12/19 students made their first search before compiling and running their code. These results suggest that students use search not only for debugging errors, as has been the focus of prior work [6, 14], but also to help select a solution approach, or refresh their memory of how to implement it. This aligns with Brandts' work on experts using the web for learning, clarifying, and remembering [4], as well as Price's work on help-seeking as problem solving [21]. Pedagogy or tools that support web search might encourage students to differentiate between these distinct help needs, since different online resources (e.g. tutorial vs documentation) may serve each better.

4.1.2 Responding to Errors. When students encountered errors, such as syntax or logical errors, they often debugged the error with web search. However, students generally responded in one of two distinct ways. Before they searched, students would engage in **Debugging Behaviors** (10/19) at least once. This included things such as mentally stepping through a loop and tracing variables (which we could observe due to the think-aloud protocol). The remaining 9 students would turn towards search and look for help without engaging in these debugging practices, suggesting that students might be looking for help without fully understanding and engaging with their erroneous code. P12 encountered a scoping error with a variable in a try/catch, briefly skimmed their entire file without looking at their initializations, and stated *"I'm not sure how I can fix the access of a variable so I'm gonna look it up."*

Students also differed in how they used error messages. Some students would **Directly Search for the Compiler Error Text** (4/19) at least once, which involved copying and pasting (or retyping) the exact error without modification, often to mixed results. P13 wrote code in order to iterate through a file, but added an extra bracket in their if statement without noticing. This resulted in the error: "`java.lang.error unresolved compilation problem`",

[2] https://docs.google.com/document/d/15iJStjBYqqJAvYfXkXXvnDs_h9tcUS2nFWz_5zSmtzs

which they searched directly, leading to a StackOverflow page detailing irrelevant information pertaining to build paths. The student expressed confusion and went back to investigating and commenting out parts of their code, before realizing the extra bracket and deleting it.

Some students used their **Compiler Error to Inform Their Search Behavior** (10/19), which involved reading the error and then reasoning about why it could possibly occur, and then translating that into a more general query, rather than copying it directly. For example P15 was trying to compare an Object-type and a String-type variable and encountered the error "`error bad operand types for binary operator '=='`", the student then thought about it, stated that they realized they needed to cast an object, and searched "`how to cast an object in an array to a double.`"

Discussion: Prior work shows that students often have difficulty understanding compiler error messages, and benefit from enhanced messages with better explanations [3]. Our results suggest that web search may serve a similar function for *some* students, especially those who can interpret the errors first, but that it is does not always answer students' questions.

4.2 RQ2: How Do Students Use What They Find?

4.2.1 Selecting a Help Resource. We found that students often move quickly to select a help resource. After entering a search query, a large majority of students, at least once, **Clicked The First Result Without Scanning** (18/19). This code applied to a median of 3 times per student. Students would enter a search query and then click the first link within a second without any obvious indication of reading or highlighting other results of the search query. In an extreme version of this we noticed that P15 would type in a query, and then *immediately move their cursor to where the first link would be before the search results even loaded so that they could instantly click on the first result.* When asked if this was a common behavior for them, they stated *"I think a lot of times like whatever Google recommends first, I usually check that first."*

After clicking into a page a large majority of students, at least once, **Looked Directly to Code** (18/19). Here, they would load a page, and then immediately scroll down to a code block, ignoring any sort of prose along the way before reading the code example. Contrast this to how some student, at least once, **Read Prose First** (10/19), and other students **Read The Surrounding Text Around Code Examples** (10/19). When asked about the above behaviors, students expressed their preference in seeing and reading code examples quickly. P17 stated, while highlighting example code in a website description: *"I'm a visual person, what I'm looking for is this right here. It gives me grounds on the kind of setup I need to have."*

4.2.2 Reliance On Search Engine Extracted Information. We analyzed how Google search specifically influenced search behavior, as Google has large majority of the global search engine market share. In our pre-interview, *all 19 students said that their preferred search engine was Google.*

In addition to actual query results, Google also returns extracted information from websites, and we noticed that a majority of students made use of **Search Engine Extracted Information** (15/19) at least once. When a query is made, Google sometimes returns a "Featured Snippet" in an effort to provide answers to searchers without requiring them to load a new page. These can include quick definitions, extracted paragraphs, or even step by step instructions. Sometimes, featured snippets would return code as well, however, this code would sometimes be malformatted or cut off due to the size of the Featured Snippet area. Featured snippets often provided quick answers for simple questions that students make. When looking up Java import syntax and found the result in a featured snippet, P14 stated that they were *"making sure I was searching for exactly what I wanted in simple terms in hope that [the featured snippet] would pop up right away. It's something that's out there on a lot of websites, so I know that it was something that would pop up fairly easily."* Google also returns a "People Also Ask" that showcases other queries that people commonly search for, which some used as basis for further search queries, or clicked on the pages provided.

4.2.3 Image Search. In our study, a small amount of students **Used Image Search** (2/19) to help search for code examples. While few, those who did use image search used it as their main search modality through the programming session, preferring to use it over the main text results page. This behavior was unexpected to all the researchers involved, so we provide a small case study using P6 to exemplify this modality of search.

During their programming session, P6 searched "`how to read a file java.`" They then went to the images tab, which showed various screenshots of Java code. They then clicked on one of the results, which expanded the image to a large enough size so they could read it. They looked at the image, before clicking on a different image and began to read it more closely, finally deciding implement the approach detailed in the screenshot. Not once did the student actually load the website the image was from; *all the work was done from within the Google Images page.*

When asked on this behavior, they mentioned that going to images maximizes the number of code examples they can see at once, and expressed that clicking links on Google results to look for examples was more time consuming. P6 stated that *"Sometimes it's easier to find the skeleton of the method that I want in images, so I usually go there first. Every time I've tried looking up for things, sometimes it's easier to find what I'm looking for in the images then in the websites where I have to scroll through everything. In the images I can just find it there, I don't have to read text."* P10 stated that *"I guess I just see a lot of examples right off the bat. I don't always do that, but it's just a good way to see a lot."*

Discussion: Li et al. found that when searching within StackOverflow, novices usually clicked the first link while experts took a more measured approach [14]. We see that this behavior extends to using the entirety of the web: a majority of the students in our study wanted code examples quickly, immediately scrolling to code within website or relying on extracted information or image search. In some cases, this behavior may be a sign of *expedient* help-seeking [19], in which students use help resources to resolve a problem quickly, without making an effort to learn. It is important to note that we did encourage students to solve problems quickly without an explicit focus on learning, and in some cases the "expedient" search results did resolve minor issues well. Compared to experts, this suggests that students may not be careful in their search for information on the web, and possibly need pedagogy or guidance on a more nuanced approach to searching.

4.3 What Barriers Do Students Face?

4.3.1 Reservations On Searching. As stated in our Methodology, it was made explicit to the students that they should search online when they are stuck. However, some students **Needed Explicit Prompting** (7/19) after struggling for an extended period of time that they could use the online resource. Within our population, we identified two main causes for students self-restricting their own usage of the internet: *Self-Reliance* and *Web Inexperience*.

Self-Reliant students stated that they liked to figure out things on their own – even self-identifying as "stubborn" in some cases. Some even stated that this behavior may not be desirable in a non-educational setting where tasks need to get done. When asked about their behavior, P12 stated that *"When I see something new, I don't like to immediately see other people's implementation, but there is a breaking point."* While they did not enjoy searching, self-reliant students were able to make use of the web for help when prompted.

This contrasts with the *Web Inexperienced* students. These students would express need for help, but even when prompted that they could use the Internet, would express that it's something they do not do often. P18 would only limit themselves to the Java API for searching, as they were taught in class. Near the end of the study session, they were spending time unproductively trying to turn a String into a Double and stated that *"If this was a problem that I ran into in class I would ask a TA for help because I'm not really sure how to do this."*, and stopped, not sure what to do next. When prompted that they could use the entirety of the Internet, not just the Java API, they would still only use the Java API. When asked, they stated *"I'm just used to how I program in class where I don't just search things up."*

4.3.2 Faulty Integrations and Compounding Errors. When integrating code examples, more students preferred to **Retype Examples** (14/19) instead of **Copy and Pasting** (7/19) from a website. Students often implemented code they find online without re-checking if their implementation was correct, occasionally leading to them running their program and encountering errors, not realizing that it stems from the code they've integrated in the past. Some students, seeing their current implementation wasn't working, would see a different way of solving the problem (such as using a different API) online and **Change Their Approach** (11/19), sometimes more than once. Some students would even search to debug *code they have integrated incorrectly*, using new code they find and end up in a spiral of **Compounding Faulty Integrations** (4/19). We take P11 as a case study for these behaviors, as they exemplify many of the failure modes that we saw across students.

P11 first searched "`how to read a csv file in java`" before writing any code. They then found an article[3] detailing multiple ways to read a CSV file, and found a solution using `Scanner` which they were familiar with. They then switched back to the IDE, and began to retype the implementation. However, while retyping, instead of typing "`new Scanner(new File(FILENAME))`" they mistakenly typed "`new Scanner(FILENAME)`", and then copied the rest of the example correctly.

After writing and compiling further non-working code, not realizing their mistake, they flipped back to the web page they retyped from and performed a high-level scan, still not noticing their mistake. They then found another website[4], which gave an example for a `BufferedReader` approach, stated that *"I'm not sure what this is"*, went back to the first article, scrolled down, and saw a simpler version of the `BufferedReader` approach. They then stated *"Instead of this approach, I'll probably set up a BufferedReader."*, marking their first approach change. They then managed to read the file in; however they struggled with exceptions. As they were typing import statements, they mistakenly attributed missing imports to why their previous code wasn't working, and made a second approach change back `Scanner`. Once realizing that wasn't their issue, they then switched approaches a third time back to the `BufferedReader` approach, and then realized they were missing exceptions.

Discussion: Much like other forms of help-seeking, students have reservations on searching, and that implicit instruction on *how* and *when* to search might be useful for students with these reservations. For those who do make use of search, sometimes they will change their approach when they are stuck, even if their code is almost correct. Research has shown that retyping code examples can beneficial for learning [9], however these behaviors suggest that retyping also allows for possible mistakes in implementation, which can cause a student to not recognize the source of an error. Pedagogy or tools could be designed to help avoid these mistakes.

5 THREATS TO VALIDITY & FUTURE WORK

One limitation is the narrow scope of the problem to achieve results in a reasonable time, which could not capture the complexity of help-seeking for something larger like a project. Another limitation is that a researcher was observing the students the entire time in order to make sure they were thinking aloud, this could possibly put stress upon the student and make them perform differently than they would be if they were programming alone. There may have been a self-selection bias within our study, where those who are willing to sign up for an participate in an extra-curricular study may not be representative of every type of student that needs help. Finally, while not a practice that is unique to just our university, the active discouragement of using the web for homework and assignments could also have influenced the behaviors and perceptions of web-search for students. However, due to the lack of qualitative research in this area, we believe that our limitations are valid and will be helpful for further studies investigating any of these behaviors in more detail.

We found students searched thought the entire process, using the web to both debug their programs as well as a problem solving method. *Can pedagogy or tools be constructed to better help with these distinct help needs?* We found students focused on viewing code examples quickly, sometimes using unexpected methods such as extracted information or image search. *Are students learning when engaging in this expedient help-seeking, and do these methods provide the same quality of content as web pages?* Finally, we found students have reservations on searching, as well as misjudge the success of their own integrations. *What factors could lead to these reservations, and how can we design pedagogy to ensure that students do not make mistakes while integrating code they find online?*

[3] https://www.javatpoint.com/how-to-read-csv-file-in-java

[4] https://www.java67.com/2015/08/how-to-load-data-from-csv-file-in-java.html

REFERENCES

[1] Mareh Fakhir Al-Sammarraie. 2017. An Empirical Investigation of Collaborative Web Search Tool on Novice's Query Behavior. (2017).

[2] Gina R Bai, Joshua Kayani, and Kathryn T Stolee. 2020. How graduate computing students search when using an unfamiliar programming language. In *Proceedings of the 28th International Conference on Program Comprehension*. 160–171.

[3] Brett A Becker, Paul Denny, Raymond Pettit, Durell Bouchard, Dennis J Bouvier, Brian Harrington, Amir Kamil, Amey Karkare, Chris McDonald, Peter-Michael Osera, et al. 2019. Compiler error messages considered unhelpful: The landscape of text-based programming error message research. *Proceedings of the working group reports on innovation and technology in computer science education* (2019), 177–210.

[4] Joel Brandt, Philip J Guo, Joel Lewenstein, Mira Dontcheva, and Scott R Klemmer. 2009. Two studies of opportunistic programming: interleaving web foraging, learning, and writing code. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 1589–1598.

[5] Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative research in psychology* 3, 2 (2006), 77–101.

[6] Preetha Chatterjee, Minji Kong, and Lori Pollock. 2020. Finding help with programming errors: An exploratory study of novice software engineers' focus in stack overflow posts. *Journal of Systems and Software* 159 (2020), 110454.

[7] Augie Doebling and Ayaan M. Kazerouni. 2021. Patterns of Academic Help-Seeking in Undergraduate Computing Students. In *Proceedings of the 21st Koli Calling International Conference on Computing Education Research* (Joensuu, Finland) *(Koli Calling '21)*. Association for Computing Machinery, New York, NY, USA, Article 13, 10 pages. https://doi.org/10.1145/3488042.3488052

[8] Yihuan Dong, Samiha Marwan, Veronica Catete, Thomas Price, and Tiffany Barnes. 2019. Defining tinkering behavior in open-ended block-based programming assignments. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education*. 1204–1210.

[9] Adam M Gaweda, Collin F Lynch, Nathan Seamon, Gabriel Silva de Oliveira, and Alay Deliwa. 2020. Typing exercises as interactive worked examples for deliberate practice in cs courses. In *Proceedings of the Twenty-Second Australasian Computing Education Conference*. 105–113.

[10] Souvick Ghosh, Manasa Rath, and Chirag Shah. 2018. Searching as Learning: Exploring Search Behavior and Learning Outcomes in Learning-Related Tasks. In *Proceedings of the 2018 Conference on Human Information Interaction q& Retrieval* (New Brunswick, NJ, USA) *(CHIIR '18)*. Association for Computing Machinery, New York, NY, USA, 22–31. https://doi.org/10.1145/3176349.3176386

[11] Carrie Grimes, Diane Tang, and Daniel Russell. 2007. Query logs alone are not enough. (2007).

[12] Andre Hora. 2021. Googling for software development: what developers search for and what they find. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. IEEE, 317–328.

[13] Stuart A Karabenick. 2004. Perceived achievement goal structure and college student help seeking. *Journal of educational psychology* 96, 3 (2004), 569.

[14] Annie Li, Madeline Endres, and Westley Weimer. 2022. Debugging with Stack Overflow: Web Search Behavior in Novice and Expert Programmers. (2022).

[15] Hongwei Li, Zhenchang Xing, Xin Peng, and Wenyun Zhao. 2013. What help do developers seek, when and how?. In *2013 20th working conference on reverse engineering (WCRE)*. IEEE, 142–151.

[16] Samiha Marwan, Anay Dombe, and Thomas W Price. 2020. Unproductive help-seeking in programming: What it is and how to address it. In *Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education*. 54–60.

[17] André N Meyer, Laura E Barton, Gail C Murphy, Thomas Zimmermann, and Thomas Fritz. 2017. The work life of developers: Activities, switches and perceived productivity. *IEEE Transactions on Software Engineering* 43, 12 (2017), 1178–1193.

[18] Silvia Muller, Monica Babes-Vroman, Mary Emenike, and Thu D Nguyen. 2020. Exploring Novice Programmers' Homework Practices: Initial Observations of Information Seeking Behaviors. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. 333–339.

[19] Sharon Nelson-Le Gall. 1981. Help-seeking: An understudied problem-solving skill in children. *Developmental review* 1, 3 (1981), 224–246.

[20] Richard S Newman. 1994. Adaptive help seeking: A strategy of self-regulated learning. (1994).

[21] Thomas W. Price, Zhongxiu Liu, Veronica Cateté, and Tiffany Barnes. 2017. Factors Influencing Students' Help-Seeking Behavior While Programming with Human and Computer Tutors. In *Proceedings of the 2017 ACM Conference on International Computing Education Research* (Tacoma, Washington, USA) *(ICER '17)*. Association for Computing Machinery, New York, NY, USA, 127–135. https://doi.org/10.1145/3105726.3106179

[22] Allison M Ryan, Paul R Pintrich, and Carol Midgley. 2001. Avoiding seeking help in the classroom: Who and why? *Educational Psychology Review* 13, 2 (2001), 93–114.

[23] Otto Seppälä, Petri Ihantola, Essi Isohanni, Juha Sorva, and Arto Vihavainen. 2015. Do we know how difficult the rainfall problem is?. In *Proceedings of the 15th Koli Calling Conference on Computing Education Research*. 87–96.

[24] Wengran Wang, Archit Kwatra, James Skripchuk, Neeloy Gomes, Alexandra Milliken, Chris Martens, Tiffany Barnes, and Thomas Price. 2021. Novices' learning barriers when using code examples in open-ended programming. In *Proceedings of the 26th ACM Conference on Innovation and Technology in Computer Science Education V. 1*. 394–400.

[25] Yuhao Wu, Shaowei Wang, Cor-Paul Bezemer, and Katsuro Inoue. 2019. How do developers utilize source code from stack overflow? *Empirical Software Engineering* 24, 2 (2019), 637–673.

[26] Xin Xia, Lingfeng Bao, David Lo, Pavneet Singh Kochhar, Ahmed E Hassan, and Zhenchang Xing. 2017. What do developers search for on the web? *Empirical Software Engineering* 22, 6 (2017), 3149–3185.