How Graduate Computing Students Search When Using an Unfamiliar Programming Language

Gina R. Bai
North Carolina State University
Raleigh, NC, USA
rbai2@ncsu.edu

Joshua Kayani*
North Carolina State University
Raleigh, NC, USA
jkayani@ncsu.edu

Kathryn T. Stolee
North Carolina State University
Raleigh, NC, USA
ktstolee@ncsu.edu

ABSTRACT

Developers and computing students are usually expected to master multiple programming languages. To learn a new language, developers often turn to online search to find information and code examples. However, insights on how learners perform code search when working with an unfamiliar language are lacking. Understanding how learners search and the challenges they encounter when using an unfamiliar language can motivate future tools and techniques to better support subsequent language learners.

Research on code search behavior typically involves monitoring developers during search activities through logs or *in situ* surveys. We conducted a study on how computing students search for code in an unfamiliar programming language with 18 graduate students working on VBA tasks in a lab environment. Our surveys explicitly asked about search success and query reformulation to gather reliable data on those metrics. By analyzing the combination of search logs and survey responses, we found that students typically search to explore APIs or find example code. Approximately 50% of queries that precede clicks on documentation or tutorials successfully solved the problem. Students frequently borrowed terms from languages with which they are familiar when searching for examples in an unfamiliar language, but term borrowing did not impede search success. Edit distances between reformulated queries and non-reformulated queries were nearly the same. These results have implications for code search research, especially on reformulation, and for research on supporting programmers when learning a new language.

CCS CONCEPTS

• **Social and professional topics → Computer science education**; • **Software and its engineering → General programming languages**; • **Information systems → Information extraction**.

KEYWORDS

Code search, Learning new programming languages, VBA

*This author performed the work while a student at North Carolina State University.

ICPC '20, October 5–6, 2020, Seoul, Republic of Korea
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-7958-8/20/05...$15.00
https://doi.org/10.1145/3387904.3389274

ACM Reference Format:
Gina R. Bai, Joshua Kayani, and Kathryn T. Stolee. 2020. How Graduate Computing Students Search When Using an Unfamiliar Programming Language. In *28th International Conference on Program Comprehension (ICPC '20), October 5–6, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3387904.3389274

1 INTRODUCTION

Software is often written using multiple programming languages [29, 51], which expects the developers to master multiple programming languages. Recent research has shown that knowledge of one language can interfere with learning a new language [41, 43]. When learning a new language, developers use an opportunistic learning strategy, relating concepts in a new language to their previous languages [43]. As the terminology between languages often differs drastically, this makes code search especially difficult.

Yet, developers frequently turning to code search to find code examples to learn from [39] and improve their productivity during development activities [5, 39, 53]. Studies on code search in software engineering [7, 8, 34, 45, 46, 48] seek to understand how and why developers search when performing their daily work [21, 27, 39, 54], introduce new code search tools or propose potential improvement of existing code search tools [3, 7, 28, 56], and suggest effective code search strategies [11, 18]. The code search studies are most often performed in the wild, so to observe developers during their normal activity.

As prior work has observed difficulties when developers search for concepts in a new language [43], we have reason to believe that code search for subsequent language learners is different than code search during normal development activities. Consider the following scenario:

A Computer Science student gets an internship at a bank. Their team manager is happy that they have several years of programming experience, and expects them to program with Visual Basic for Applications (VBA) and create macros in spreadsheets. However, the student has never been taught VBA at school, and now they have to learn this new language on their own.

The intern gets their first task, and decides to search online for how to perform it in VBA, but the keywords are specific to the language they know, such as "dictionary" from Java paired with their new context, VBA. This leads to results that are either in VBA but with information not relevant to the task at hand, or results in Java that are relevant, but in the wrong language. The intern wonders, "*is there any way to improve the search success? What online sources should be consulted?*"

In this scenario, the intern cannot find the resources they want due to mismatched terminology between the language they know, Java, and the language they need, VBA. This common situation is observed in prior research [42]; in this exploratory work on code search, we target students learning a new language.

Understanding and addressing the challenges learners encounter during search could lead to better search experiences for learners, especially students, who have less experience than the professional developers with learning a subsequent language. Therefore, we design a study that involves tasks in a language unfamiliar to the participants and recruit graduate students in Computer Science as participants.

Our methodology involves logging search and browser activities and periodically surveying participants about their current tasks, similar to prior work [39]. However, unlike prior work, we specifically explore the factors that lead to search success (RQ3), which is determined by explicitly asking participants if a search was successful. Prior human studies with code search do not indicate what factors lead to successful searches, in large part because obtaining evidence of search success is tricky. Relying on result clicks [14] is incomplete as often the answer appears in the preview accompanying a search result. Relying on evidence of query reformulation can reliably find failed searches [20, 24], but not successful ones. We address this shortcoming using *in situ* surveys that specifically ask about search success.

Given the similarity in methodology, where appropriate, parallels are drawn between the learners in our study and professional developers from a similar study at Google [39]. While the contexts of the studies varied widely (i.e., normal developer workflow [39] with Google Developers vs. a lab study with students working with an unfamiliar language), similarities and differences shed light on how we may be able to train subsequent language learners to use search better in their workflow, or build a better search tool to facilitate more successful searches. We summarize our contributions as follows:

- insights on the factors that make a query successful,
- better understanding of the code search behaviors of subsequent language learners, and the similarities and differences of search behaviors between learners and professionals, and
- suggestions on improving the success of searches when learners are working with an unfamiliar language.

To our knowledge, we are the first to study how subsequent language learners [6, 19, 32, 41] perform code search, and our results have implications for how to better support them. Our main findings and suggestions include:

- Successful searches use natural language phrases, such as "if statement," instead of coding shortcuts, such as "if".
- Successful searches more often consult official documentation and tutorials with example code rather than Q&A sites.
- Reformulations occur much more quickly after a previous query than searching for a new topic. The Levenshtein distance between successive queries alone may not indicate a reformulation.
- As term borrowing was commonly observed in search queries, an "API translator" that can map APIs across languages would assist subsequent programming language learners.
- Since learners often search to explore language syntax, APIs/libraries, and example code, we encourage CS educators to include short sample code when presenting or requiring use of new APIs/libraries.

This paper describes the context and methods we used to collect the surveys and logs for this study in Section 2. Section 3 presents the detailed study results. Implications of the findings are suggested in Section 4.2 and the potential threats to validity are discussed in section 4.4. Finally, Section 6 presents some concluding remarks and suggests modifications on future work.

2 STUDY

We frame this study around the following three research questions:

RQ1: Why do subsequent language learners search?
RQ2: What does a typical search session entail for a subsequent language learner?
RQ3: What are the factors that impact the success of search queries for subsequent language learners?

2.1 Study Design

Using a combination of surveys and logs, Sadowski, et al. [39] explored the search behavior of Google developers; it is from this study that we derive our methodology. Table 1 summarizes the similarities and differences of the context and collected data between the previous study and this study. To target subsequent language learners, we run our study in a lab environment with graduate students. Participants were given tasks in the unfamiliar languages. Due to the experimental context, we were able to collect more survey responses per query as compared to the previous study, providing a higher density of auxiliary information per query.

One of the shortcomings of the original study is that there was no notion of search success in the logs or the queries. That is, it was not clear from the data when a search session was successful or unsuccessful. To address that shortcoming, in this study, we ask explicitly about search success in two surveys, one deployed when a tab is closed (Figure 4) and another deployed when it is suspected a query is reformulated (Figure 3). Another shortcoming is the use of edit distance to detect reformulation, which may not represent actual reformulation; we address this by asking participants whether their current query is intended to solve the same problem as the previous query in a *reformulation survey* (Figure 3).

2.2 VBA Programming Tasks

We chose Visual Basic for Applications (VBA) because it is a popular programming language with many online resources [4, 13], but it is not commonly used by the graduate students in Computer Science at North Carolina State University (NCSU, our participant population). Each task includes a general learning goal (e.g., conditions and loops, string manipulation), a task description with requirements, and a sample output/result for the task. The five tasks[1] are summarized as follows:

Task 1 provides participants a list of numbers in a column, and asks participants to record a macro and create a button that returns the average value of the numbers in this column

[1] Artifacts are available at: https://github.com/ginaBai/CodeSearch_VBA

Table 1: Summary on Study Designs of the Original Study [39] and this Study

	The Original Study	This Study
Duration	15 consecutive days (4 weekends and 11 weekdays)	90-minute lab session
# Participants	27 Google developers	18 NCSU graduate students
Compensation	None	Drawing to win a $30 Amazon gift card
Context	Daily work	5 VBA tasks
Survey Types	1 survey, 4 questions	4 surveys, 2-6 questions each (Section 2.3.3)
Survey Setting	10 surveys per day, 10-minute time interval	10 surveys per hour, 1-minute time interval
Survey Responses	394 responses	216 responses
Survey Collection	Browser extension, surveys triggered when developers accessed the internal code search site	Browser extension, survey triggered under four conditions (Section 2.3.3)
# Total Log Entries	77,632 entries	3,508 entries
# Total Queries	3,870 queries	229 queries

when the button is clicked. This task can be completed with or without VBA code.

Task 2 requires participants to create a button that creates a message box with "Hello World!" on it once clicked.

Task 3 provides participants a list of scores in a column called *Grade*, ranging from 43 to 100, and ask participants to create a button that populates a column called *P/F* with P or F for Pass or Fail, respectively, depending on *Grade*: P is for a score that is greater than or equal to 60 in column *Grade*, and F otherwise.

Task 4 provides participants an unsorted inventory list of unique labels in a column called *Item* with their prices listed in a column called *Price*. Their goal is, given a separate target list of labels, to create a button that populates a column *Price* for the items in the target list based on the inventory list. The Excel built-in function `VLOOKUP` is not allowed in this task.

Task 5 provides participants a list of phone numbers in the format of *(XXX)XXX-XXXX*, and asks participants to create a button that populates a column called *Area Code* with unique area codes and their associated counts in a column called *Count*.

Participants were instructed to complete the tasks in order.

2.3 Data Collection

We developed a Google Chrome extension[2] that records search events and browsing history and deploys surveys. To capture the questions the participants are trying to solve, we prompt users with short surveys periodically. In this way, we are able to combine the survey responses with logs analysis.

2.3.1 Tool Implementation. The implementation consists of two parts: a client-side Google Chrome browser extension for logging data and a Flask web server for storing data. On the client side, the browser extension does the following: 1) initialize browser local storage that holds the log information, including issuing a unique 10-digit ID number to each participant, 2) track Google searches, 3) track link clicks on all pages, 4) detect possible query reformulations via word-level Levenshtein distance (if distance ≥ two words [23]), 5) format and deploy the surveys, and 6) send all collected logs to our server for storage. All collected data are stored in a password-secured SQLite database on an encrypted server.

2.3.2 Procedure. The study was conducted in a lab setting over two sessions, 90 minutes each. Participants attended one lab session only. Participants were instructed to install the Google Chrome extension on their personal laptops and keep the extension enabled during the lab session.

2.3.3 Surveys. To avoid over-taxing the participants, a maximum of 10 surveys per hour were deployed with a one-minute minimal interval between surveys. A survey is triggered when a participant performs one of the following four actions:

(1) **Finishes installing the extension**
Preliminary Survey (Figure 1). We collect participants' technical background such as programming experience, their intentions of code search, and demographic information.

(2) **Makes a search**
Context Survey (Figure 2). We ask participants about the activities they are doing, and the specific question they are solving. Questions Q2, Q3, and Q5 are from prior work [39]. In addition, we ask what approach(es) they've adopted to solve the tasks.

(3) **Reformulates a query**
Query Reformulation Survey (Figure 3). A query is identified as a potential reformulation if its word-level Levenshtein distance from the prior query is greater than or equal to two. This survey verifies the reformulation by asking in the current query is related to the previous one. It also asks if the prior query solved the problem. If so, how did it help; if not, what information is missing.

(4) **Closes the search tab**
Search Tab Closed Survey (Figure 4). This survey asks if the prior query solved the problem. If so, how did it help; if not, what information is missing.

A survey will not be triggered by any event if the prior survey is deployed within a minute or the number of deployed surveys has reached the hourly limit.

We asked participants to complete the survey every time it is deployed, although completion is not forced.

[2]Chrome web store: https://chrome.google.com/webstore/detail/chrome-code-search/ljdehfmdnkelbnpogdeobpaldeibecek

Q1: How would you rate your overall programming experience?
A. Novice B. Intermediate C. Expert

Q2: What languages do you work with most frequently?
☐ Java ☐ C ☐ C++ ☐ Python ☐ C# ☐ PHP ☐ JavaScript
☐ Visual Basic .Net ☐ Perl ☐ Assembly ☐ Ruby ☐ Delphi
☐ Object Pascal ☐ Swift ☐ Objective C ☐ MatLab ☐ R
☐ PL/SQL ☐ COBOL ☐ Groovy ☐ Other: ____________

Q3: In what contexts or for what purpose(s) do you search for codes via websites during software development activities?
Short Answer: ________________________

Q4: When solving a programming/development/technical problem, how often do you: (rank the following from most often to least often with number 1, 2, and 3)

Ask a colleague/friend	A. 1	B. 2	C. 3
Web or online search	A. 1	B. 2	C. 3
IDE search/exploration	A. 1	B. 2	C. 3

Other: ____________

Q5: What's your occupation?
A. Undergraduate Student B. Graduate Student
C. Student with working experience in the industry
F. Other: ____________

Q6: How would you identify your gender?
A. Female B. Male C. Non-binary D. Prefer not to disclose

Figure 1: Preliminary Survey

Q1. What languages is your search related to? (separate each language with a newline [hitting the enter key]) ______________

Q2: What are you doing?
A. Exploring (Looking into a certain programming language, etc.)
B. Debugging code (Why does ABC return null?, How to use XYZ debugger?, etc.)
C. Designing/Developing a feature D. Maintaining old code
E. Other: ____________

Q3: How familiar are you with the code you're looking for?
A. Very Familiar (e.g., I've written something similar in the past)
B. Somewhat Familiar (e.g., I've seen something like it before)
C. Not familiar (e.g., this is entirely new to me)

Q4: What else have you tried to answer this question?
☐ Searched local codebase
☐ Consulting a friend/colleague in person
☐ Consulting a friend/colleague via email, IM, etc.
☐ Consulted a book/textbook

Q5: What question are you trying to answer? ______________

Q6: What type of code are you working on?
A. Backend (database, cloud storage, etc.) B. Front-end GUI
C. Middleware D. Other: ____________

Figure 2: Context Survey

Q1: When you searched, [Previous Query], did you solve the problem?
A. Yes (jump to Q2_y) B. No (jump to Q2_n)

Q2_y: What solved it?
☐ Found a needed API (how to modify CSS with jQuery)
☐ Found a needed implementation (bubble sort implementation in Java)
☐ Asked a colleague in person ☐ Asked a colleague online/email
☐ Other method: ____________
Q2_n: Is your search, [Current Query], related to the problem?
A. Yes (jump to Q3) B. No

Q3: What do you need to solve the earlier query?___________

Figure 3: Query Reformulation Survey

Q1: When you searched, [Current Query], did you solve the problem?
A. Yes (jump to Q2_y) B. No (jump to Q2_n)

Q2_y: What solved it?
☐ Found a needed API (how to modify CSS with jQuery)
☐ Found a needed implementation (bubble sort implementation in Java)
☐ Asked a colleague in person ☐ Asked a colleague online/email
☐ Other method: ____________
Q2_n: Why?
A. Need more related information B. Need to reformulate the query
C. Other: ____________

Figure 4: Search Tab Closed Survey

2.3.4 *Log Data.*

A log entry is created when a participant performs one of the following trigger events: searches, closes a tab, clicks a link, switches tabs, or switches to another application. The logged information includes:

- **UserID**: A unique number that distinguishes participants.
- **Time**: Time of this activity.
- **Type**: Types of browser activities, including: closed the tab, clicked a link, deployed survey, received survey responses.
- **URL**: The url of the website the participant clicked.
- **TabID**: Browser-assigned ID for each tab to track the tab activities.
- **Query**: String from an online search query.

2.4 Participants

We invited students enrolled in a graduate software engineering course at North Carolina State University via email to participate in the study (response rate: 42.6%). A short introduction to this study, including the purpose thereof, and the tool (Google Chrome extension) used in the study, was made available to all potential participants. As compensation, participants who have the extension enabled and recording for at least 45 minutes were eligible to enter into a drawing to win a $30 Amazon gift card. In total 23 students completed the initial surveys. After discarding data from five participants whose interactions were recorded for less than ten minutes, we were left with data from the remaining 18 participants for analysis (average: 78.5 minutes, median: 84 minutes), which includes four female students and fourteen male students.

The 18 participants self-reported their expertise in general programming skills: one as a novice developer, five as experts, and twelve as intermediates. Two participants claimed to have prior working experience in the industry. All participants often search on websites for code, and the majority of participants sometimes or often ask friends for help (16/18) or search on IDEs (16/18). Participants came from a diverse programming language background, including object-oriented programming languages and scripting languages, but not VBA. The top five frequent programming languages used by the participants were: Java, JavaScript, C++, C, and Python.

2.5 Analysis

We grouped the logs by participants' IDs and sorted them by log time. Surveys were tabulated and associated with browser events for analysis. Logs were analyzed to collect 1) search queries, 2) ordered list of clicked website URLs associated to each query, and 3) time spent by a participant on a clicked result page.

The logs were split into search sessions. Due to our study context, adopting the definition of a search session from the study on Google developers [39] was problematic. When considering a search session as a series of developer activities separated by 6-minutes of inactivity, each participant in our study had exactly one search session. Thus, we treat each search query and its associating result clicks as one search session (referred to as "micro-session" in prior work [39]).

Our reformulation surveys were deployed when the Levenshtein distance was two or greater, in line with prior work [23, 34]. If participants answer *yes* to $Q1$ (Figure 3), this indicates that the current query is not a reformulation and begins a new topic. If the participant answers *no* to $Q1$ and *yes* to $Q2\_n$, this means that the previous query did not solve the question and the current query is related to the same problem, and thus the current query is a reformulation. If a participant answers *no* to $Q1$ and *no* to $Q2\_n$, we assume that the problem is the same but the participant is trying a new approach, and thus the query is not a reformulation. While prior work makes assumptions about reformulation based on topic analysis or query distances, these survey responses serve as a ground truth for reformulation analysis. While we do not have enough data to build a classifier to predict reformulation, we can analyze the context around known reformulations versus known non-reformulations.

2.6 Data Summary

The 18 participants generated 3,508 log entries, including 216 survey responses and 229 queries during the lab sessions. Each participant made an average of 12.7 queries (median: 13 queries), each query led to an average of three search result clicks (median: two clicks), and the average time interval between each query was 5.8 minutes (median: 4.3 minutes).

As to the surveys, 216 of 389 deployed surveys were completed and collected, including 18 *Preliminary Surveys* (response rate: 100.0%), 35 *Context Surveys* (response rate: 39.3%), 138 *Query Reformulation Surveys* (response rate: 54.5%) and 25 *Search Tab Closed Surveys* (response rate: 86.2%).

3 RESULTS

In this section, we present the results for each research question in turn.

Table 2: Answers to 18 Preliminary Survey Question "In what contexts or for what purpose(s) do you search for codes via website during software development activities" (*Category* - Multiple Answers are Allowed)

Category	#par (%)
Look for Example Code (How)	
Library / Class / API	9 (50.0%)
Method / Function / Operation	7 (38.9%)
Syntax	9 (50.0%)
Solution	4 (22.2%)
Explore Code (What)	
Concept	3 (16.7%)
Determine Reasons (Why)	
Debug	7 (38.9%)

Table 3: Answers to 35 Context Survey Question "What are you doing?" (*Goal* - Multiple Choice) and Question "What are you trying to answer?" (*Example* - Short Answer)

Goal	Example	Num
Exploring	"How to enable developer tools in Excel" "How to do lookup in excel" "How to print in VBA"	19
Designing	"Basic steps to record an excel macro" "Using and iterating hashtable in VBA" "Find unique values in a column"	8
Debugging	"Loop over a column excel vba" "Create dictionary in excel vba" "Found an implementation but was getting error"	6
Other	"enabling a feature in Microsoft Excel"	2

3.1 RQ1: Why do subsequent language learners search?

Learners most frequently searched for example code and potential ways to resolve bugs or errors in their programs.

We analyzed the self-reported purposes of why participants search during programming. In 18 *Preliminary Surveys*, the most frequently reported goals were to find specific syntax, libraries/-classes/APIs, implementations or sample usage of methods/functions/operations during development. Table 2 shows three main reasons why participants search (*Category*), where the categories were adopted from the prior work of Sadowski, et al [39], and the number of participants who stated these reasons and its percentage out of 18 (*#par (%)*). For example, nine participants claimed that they usually search for correct *syntax* during software development; an example query made by a participant is, *"vba if statement"*.

We then explored the questions participants were trying to solve as collected in 35 *Context Surveys* (See Table 3). Participants were asked to categorize their motivation into exploring, designing or debugging (*Goal*), and then answer the open-ended question, "What question are you trying to answer?" with a brief description of current search activity; example queries can be found in column *Example*. For example, 19 queries were made to explore how to perform an action in Excel or with VBA with queries such as *"How to enable developer tools in Excel"*.

Table 4: Search Sessions and Corresponding Purpose of Search Reported in Context Surveys

Search Session	Description	#Session	Purpose of Search	#Purpose
S	1 Search Query + 0 Result Click	34 (14.8%)	Exploring	4 (11.4%)
			Designing	3 (8.6%)
			Debugging	2 (5.7%)
SC	1 Search Query + 1 Result Click	68 (29.7%)	Exploring	6 (17.1%)
			Designing	1 (2.9%)
			Debugging	1 (2.9%)
SCC+	1 Search Query + 2 or more Result Clicks	127 (55.5%)	Exploring	11 (31.4%)
			Designing	4 (11.4%)
			Debugging	3 (8.6%)
Total		229 (100.0%)		35 (100.0%)

Participants explored code (Table 3) and searched for example code (Table 2) frequently. They also searched for assisting debugging, which matches their self-reported purposes for searching were example codes, including APIs, syntax and existing solutions, and determine reasons why code did not work correctly (Table 2). This observation also matches the search goals of professional developers reported in the prior work [39]. While only 10% of professionals' queries tried to determine why something is failing, 17% of students' queries recorded in *Context Surveys* focused on debugging.

3.2 RQ2: What does a typical search session entail?

Each query was followed by three result clicks on average. An average participant spent about five minutes before reformulating a query. Learners generated more verbose queries than professionals did (5.6 words vs. 1.9 words) and required longer time to scan the search results.

We identified three typical search sessions (*Search Sessions*), as shown in Table 4. The column *Description* introduces the session formats, and the column *#Session* introduces the frequency and their percentage among all search sessions. We adopted a truncated regex-like syntax for the sessions: where *S* is a search query and *C* is a result click. For example, *SCC+* means one search query followed by two or more result clicks. We also investigated the *Context Surveys* within each session. The majority of SCC+ patterned search sessions were exploring new content/topics (11 in *#Purpose*). Even when a search is not followed by a click, as in the *S* pattern, exploring is the dominant purpose of the search, representing 44.4% (4/9) of the *S* sessions with surveys. This could happen when the previews on the search results contain enough information for participants to find what they need.

As for search query length considering all 229 queries, an average query contains 5.6 words, with a median of five words and a maximum of 17. These numbers are higher than the numbers reported in prior work, which were 4.2 words per query [44] and 1.9 words per query [39]. The increment in words may be caused by participants' unfamiliarity with the programming language used in the study, or from the fact that they were using general web search like [4] and [34], which also report long code-related query lengths.

Focusing in on reformulations, we examine the queries that trigger a reformulation survey. The query that triggers the survey is the *current* query and the one before it is the *previous* query. Of the 138 queries that triggered a reformulation survey, 48 (34.8%) were reformulations (answering *no* to $Q1$ and *yes* to $Q2\_n$ in Figure 3) while 90 (65.2%) were not (answering either *yes* to $Q1$ (86 surveys) or *no* to $Q1$ and *no* to $Q2\_n$ (4 surveys) in Figure 3).

When the current query is a reformulation, this occurs after an average of 3.0 (median = 2.0) result clicks; current queries that are not reformulations occur after 3.0 result clicks (median = 1.5). Based on the medians, this shows a slight uptick in clicks preceding a reformulation. The average time between a previous and current query is 4.7 minutes (median = 3.2) for a reformulation and 8.3 minutes (median = 6.0) for a non-reformulation. This indicates that reformulations occur much more quickly after a previous query and that less time is spent on each result click. However, a typical professional developers needed only eight seconds to reformulate a query [39]. This difference is possibly due to our participants' unfamiliarity with VBA, with the professionals' strong familiarity with their context, or with differences in how reformulations were computed in this analysis.[3]

Considering query length, when a query is reformulated, the length of the query is sometimes modified substantially (e.g., removing 12 words or adding seven words), and sometimes modified very little (e.g., changing one word, adding a word). The Levenshtein distance between a reformulated query and the previous query is 5.3 (median = 4.5) and the average distance between non-reformulated queries is similar, 5.3 (median = 5.0). These data provide preliminary evidence that basing reformulation on query distance metrics alone may not yield accurate results.

3.3 RQ3: What are the factors that impact the success of search queries?

When looking for APIs or implementations, 71 of 107 (66.4%) searches were successful. In addition, consulting documentation and tutorials led to more successful searches than Q&A sites (e.g., StackOverflow). Learners frequently borrowed terms from languages with which they are familiar when composing queries, these queries were more successful on average than the typical query.

[3]Prior work [39] used a word-level Levenshtein distance of at most one to identify reformulation.

Table 5: 107 Successful Queries and Corresponding Reasons for Success (86 from *Reformulation Survey* & 25 from *Search Tab Closed Survey*)

Problem were solved by...	#Queries
Finding relevant API	47 (43.9%)
Finding relevant implementation	53 (49.5%)
Consulting others in person	4 (3.7%)
Other	3 (2.8%)
Total	107 (100.0%)

Table 6: Online Sources & Success

Online Sources	#Succ	#Fail	Total
Q&A sites only	19 (11.7%)	9 (5.5%)	28 (17.2%)
D&T sites only	54 (33.1%)	14 (8.6%)	68 (41.7%)
Both Q&A and D&T	25 (15.3%)	20 (12.3%)	45 (27.6%)
None	9 (5.5%)	13 (8.0%)	22 (13.5%)
Total	107 (65.6%)	56 (34.4%)	163 (100.0%)

We consider a search query to be successful when a participant selects option *Yes* to the question "*When you searched, [Previous/Current Query], did you solve the problem?*" in either the *Query Reformulation Survey* or the *Search Tab Closed Survey*. In total, we collected 138 *Query Reformulation Surveys* (86 successful & 52 unsuccessful) and 25 *Search Tab Closed Surveys* (21 successful & 4 unsuccessful).

Among 107 surveys associated with the successful searches (Table 5), 47 surveys reported that problems were solved by a search result that contained a relevant API, 53 surveys reported that problems were solved by finding relevant implementations, four searches were concluded when the participant consulted others in person, and two problems were resolved by the participants themselves, while one did not provide detailed information as to why the search was successful. Among 56 surveys associated with unsuccessful searches, only six responses on the reasons of unsuccessful queries were submitted; the other 50 did not provide a reason for failure. Of the six, three stated that more information is needed, two stated that the query reformulation is needed, and one stated that specific functions or API is missing from the search results, which prevented this search from solving the problem.

We looked into online information sources consulted by participants and manually classified the sources into two categories: 1) Q&A sites (e.g., Stack Overflow, forums), 2) official documentation and official/third-party tutorials sites (D&T sites), similar to prior work [2, 8]. Table 6 shows the type and the number of clicked search results from each query. Table 6 lists the categories (*Online Sources*), their associated successful or unsuccessful queries (*#Succ, #Fail*). We observed that among successful queries, 50.5% (54/107) were followed by only documentation and tutorials sites, making that the most common resource leading to search success. Similarly, of the queries that consulted only D&T sites, 79% (54/68) were successful. This finding matches the prior work of Bai, et al. that consulting documentation and tutorials only are most likely to succeed during the search [2].

We also identified that 16/18 participants borrowed the terms from languages with which they are familiar. Similar borrowing behavior was also reported in prior work [1, 43], and developers claim that previous knowledge supports retrieval of new information; our results concur. We found 65 queries used terms such as "dictionary", "hashtable", "hashmap", "regex" from languages like Java and C++. However, this term borrowing sometimes introduced term mismatch, that is the cases when vocabulary mismatch between queries and documents (e.g., [10, 15, 55, 58]). For example, borrowing the term "dictionary" from Java. When five participants intended to search for functions in VBA that behave like `java.util.Dictionary`, which should be the `VLOOKUP` function, they searched for, "*dictionary in excel*", "*add or edit words in a spell check dictionary*" or, "*creating a dictionary with Microsoft Excel*".

The impact of term mismatch in terms of query success and reformulation is fairly neutral. Queries with borrowed terms were more successful, on average; this observation is further supported by a result in general information search [57] that finds engineering and science students who had higher levels of domain knowledge retrieved more relevant documents for the search question. However, these queries were no more associated with reformulation than any other query. For the 65 queries demonstrating term mismatch, 34 (52%, 26 from *Reformulation Survey*, 8 from *Search Tab Closed Survey*) were associated with a success survey, and only 15 (23%, 13 from *Reformulation Survey*, 2 from *Search Tab Closed Survey*) were reported unsuccessful. Despite the term mismatch, these queries were more successful on average than the typical query (77% success vs. 66% success, per Table 6). We conjecture that this is because VBA is so well documented that the current search algorithms are able to handle the term mismatch; this may not be the case with newer or less common programming languages.

There were subsequent query reformulations for 33% (13/39) of the term borrowing queries, where 39 of the term borrowing queries were followed by a reformulation survey, and 13 of the responses explicitly indicated reformulation occurred. This is the same proportion we saw from all reformulation surveys (34.8% are reformulations, 48/138).

Anecdotally, we observed there was a higher occurrence of the phrase, "*how to*" in successful queries than unsuccessful queries (12.1% vs. 5.4%). Over 80% (13/16) of "*How to*" queries successfully supplied participants with the needed information. As an example, the query, "*how to create a button and assign a macro in excel*" succeeded in finding the desired information, whereas the query, "*assign macro to a cell vba*" failed. The completeness of phrases in the queries may also influence the search result. For example, while four queries contained the complete terminology "*if statement*" instead of "*if*" in successful queries, there was no "*if statement*" observed in any of the unsuccessful queries. We also found a case where an unsuccessful query used "`==`" instead of spelling out "*equal to*" like a successful query did. More exploration with a larger set of successful and unsuccessful queries may shed more light on which phrases lead to higher success in the searches.

Also noteworthy is that participants were allowed to consult others in person during the study. We observed that the four problems involving consultation with other people were successfully solved. Future work should look at the interplay between search and discussion.

Table 7: Summary on Findings of The Original Study [39] and This Study

Professionals [39]	**Subsequent Language Learners**
Why do they search?	
RQ1: Developers use code search to answer questions a wide range of topics, including how to do something, learning what code does, and determining why code is behaving as it is.	RQ1: Most searches are associated to goals of exploring for example code and designing a new feature. Subsequent language learners also search for debugging.
What are they looking for?	
RQ2: Developers search for examples more than anything else. Developers navigate code search tools even through code they know well.	RQ1: Subsequent language learners search for example code more than anything else. All participants search for exploring regardless of the familiarity to the code.
Query properties	
RQ3: The average length of the queries was 1.85 words. Most queries were scoped to a subset of the code repository.	RQ2: The average length of the queries is 5.6 words. Most queries contain keywords “VBA” and “excel” to restrict the search.
RQ3: Nearly a third of searches are incrementally performed through query reformulation.	RQ2: Only 35% (48/138) of the suspected reformulations were actual reformulations. Adopting the reformulation definition from the original study, we found only 5.2% (12/229) of the queries had an edit distance of one.
Time between queries	
RQ3: A typical professional developer requires eight seconds (average of 23 seconds) to scan the search results, determine if they meet their needs, and reformulate an initial query.	RQ2: A typical subsequent language learner requires three minutes (average of five minutes) to scan the search results, determine if they meet their needs, and reformulate an initial query.
Search frequency	
RQ4: An average professional developer composes 12 search queries per weekday.	RQ2: An average subsequent language learners compose 12.7 queries in a 90-minute lab session.
Clicks per query	
RQ4: Each query lead to 1.3 clicks on average. Most micro-sessions contain one search without any clicks.	RQ2: Each query lead to three clicks on average. Most search sessions contain one search followed by at least two clicks.
Clicks vs. Search goal	
RQ5: Most searches on code reviews (determine why) lead to multiple clicks. Developers also make multiple searches and clicks when exploring for example code and designing.	RQ2: Most searches on exploring new content/topics and all debugging search activities led to one search followed by multiple clicks
Clicks vs. Code familiarity	
RQ5: Code familiarity does not mean less clicks.	RQ2: Code familiarity does not mean less clicks.
Search success	
N/A	RQ3: Learners frequently borrow terms from familiar languages, and hence be more successful than the typical query. The most successful searches occur when looking for APIs or implementations.

4 DISCUSSION

In this section, we compare the student search behavior to professionals, provide the suggestions to improve search success, and discuss the potential threats to validity.

4.1 Comparison of search behaviors between professional developers and subsequent language learners

Our study can be viewed as a partial replication of the original study methodology [39] with a drastically different context. In our study, we investigated how subsequent language learners search for code in an unfamiliar programming language, rather than using professional developers conducting their daily work. Like the original study, we explored the reasons why participants search, the properties of search queries, and search sessions in micro level. To extend the original study [39], we analyzed the properties of successful queries with the survey responses collected right after either a search activity or a search tab being closed. We also explicitly ask about query reformulation instead of speculating based on query edit distance. Table 7 summarized the similarities and differences of the main findings of the original study and this study.

For example, search sessions observed in this study were more likely to have at least two clicks after one search activity (55%), while the professionals click no result in the most search sessions (38.8%). While it only took professionals 23 seconds to reformulate or make a new query, learners usually spent five minutes before reformulating or starting a new query. Learners searched more frequently than the professionals did (13 queries/hr vs. 12 queries/day).

4.2 Successful Search Behaviors for Subsequent Language Learners

Search success in this study was associated with consulting official documentation and tutorials rather than on Q&A sites (e.g., Stack Overflow). Additionally, mapping the context in an unfamiliar languages to previous knowledge in a familiar language, and borrowing the corresponding terms, also led to search success. Lastly, natural language phrases were more associated with success than coding shortcuts. For example, spelling out "equals" instead of "==", and search with "if statement" instead of "if". These behaviors may increase the likelihood of finding relevant results, at least in contexts similar to the one studied here.

4.3 Implications

Our results have implcations for future work in the following research areas:

4.3.1 Debugging with Spreadsheets and Search. Search has become an important process during debugging tasks [39]. Prior work on debugging spreadsheets has shown that end-user programmers use eight strategies when debugging in spreadsheets [50]. These are: dataflow, testing, code inspection, specification checking, color following, to-do listing, fixing formulas, and spatial. However, code search was not part of the equation. In our study, all debugging-related search queries aimed to fix the formulas. This is likely due to the design of the study, which aimed to explore how participants use search for learning a new language, and is not reflective of general debugging strategies used in spreadsheets. However, future work should explore the interplay between search and the spreadsheet debugging strategies, especially in the context of larger tasks.

4.3.2 Query Reformulation with Domain Knowledge. Various strategies for reformulation have been adopted by researchers, such as a small edit distance [39] or a potentially large edit distance [23, 34]. In this work, we trigger a reformulation survey when the distance between a current query and the previous query is at least two, measured using Levenshtein distance. The survey responses provide a ground truth from participants about whether or not they are performing a reformulation. What we find is that of the 138 reformulation survey responses, 90 explicitly state that they were not reformulating, showing a 65.2% false positive rate for this approach to reformulation detection. In prior work [39], a Levenshtein distance of at most one was used to identify reformulation. In their dataset, nearly one third of the searches were flagged as reformulation, whereas in our study only 5.2% were flagged as reformulation using that same approach (Table 7). When we looked at distances between non-reformulated and reformulated queries (Section 3.2), we find that the average edit distances are similar, making it difficult to use edit distance alone to indicate reformulation.

Reformulation detection based on edit distance alone seems suspect. Still, we find that approximately 1/3 of the queries that triggered the reformulation survey were actual reformulations. This frequency of manual reformulation motivates other efforts by software engineering researchers to automatically reformulate queries based on query properties, hence reducing developer efforts [15].

4.3.3 API Translation. Shrestha, et al. [43] point out that developers usually face a selection barrier [26] when moving to a new programming language, and searching for the right terminology and code example is difficult. They also concluded that learning a language is difficult when there is little to no mapping of features to previous languages [43]. While in this study term mismatch was not an issue for developers in that reformulation was not more likely and search success was actually higher, we conjecture that this was due to the study context. VBA was chosen because it is well-documented, and the participants seemed to benefit from this. For languages or APIs that are less well documented, we envision "API translators" that can map an API in one language to a destination language and provide short sample code would assist developers to work and learn more efficiently, especially when migrating languages, or working with new, unfamiliar languages. Some of these features are present in tools like Mica [49], which can assist developers in finding the right API classes and methods given a description of the desired functionality. Chen, et al. [9] also present an approach that can recommend analogical libraries for different programming languages or different mobile platforms given a library.

4.4 Threats to Validity

Participants knew their searching and browsing activities were being recorded, and were periodically surveyed, which may have influenced their behavior.

Some metrics used in this study may not be consistent across participants, such as #avgClick/Query and #avgQuery, since the search results and available documents may vary due to the personalized search results. In addition, the Chrome extension did not record if a search query was a result of using an auto-complete query suggestion. Future studies should enforce non-personal results and detect auto-complete.

Participants were required to work on these tasks in a lab environment in a limited time, which could potentially impact their search behaviors. Response bias may be introduced in the participant self-reported surveys, and impacted the validity of surveys.

All participants were graduate students and the majority claimed that their overall programming skills ranged from intermediate to expert (17/18). Therefore, the results may not generalize to other populations. A replication with a more diverse and larger set of students is needed.

We use survey responses on the reformulation survey as ground truth for reformulations. However, participants may have different notions about what is and is not reformulation, which may impact the validity of the ground truth.

The conclusions were drawn based on our participants' search behaviors when using VBA, an event-driven programming language used by Excel, which may not generalize to learning other languages.

5 RELATED WORK

We focus on related work that investigates code search behaviors, explores end-users' engagement with spreadsheets, and discusses the learning barriers in programming systems.

5.1 Code Search

As with our study, research on general information search has also involved student subjects (e.g., [52, 57]). Of particular relevance, one study found that university students perceived study-oriented topics as harder to search for than daily life information [52]. Prior work on code search has yielded similar results, finding that code search tasks take more effort than information search tasks [34].

Various studies focusing specifically on code search have been performed by surveying participants about the reasons why they searched [21, 39, 45, 54], the tools they searched with [46], the popular search sites [48], as well as their selection criteria for code [27]. Different data analysis strategies are also adopted, for example, some studies focused on directly observing participants' behaviors [47], while some other studies focused on collecting and analyzing the logs, such as works done by Brandt, et al. [7, 8], with participant solving the pre-selected tasks. Sadowski and colleagues [39] also combined surveys and logs to analyze developers' code search behavior when working on daily tasks.

Automatic API recommendation and query reformulations are also being studied. Rahman, et al. proposed a novel query reformulation technique, which can translate a natural language code search query into a ranked list of relevant Java APIs [35, 36]. Various strategies for reformulation have been adopted by researchers, such as a small edit distance [39] or a potentially large edit distance [23, 34]. In this work, we explicitly ask participants about whether or not their query is a reformulation in an effort to avoid issues around automated detection of query reformulation.

Researchers [4, 34] specifically explored search sessions to learn the usage of the web search by software engineers. They concluded that the major intentions of web search are finding information to 1) debug an error or an issue, 2) accomplish a specific task, 3) learn about a topic and 4) learn about a specific API element. The researchers also suggested that code related queries are more verbose [34]. Code related queries also lead to higher rates of reformulation, longer dwell time, and fewer clicks [4, 34].

Various code search tools have been developed to facilitate developers with searching for the desired code, such as Koders, Krugle, and Sourcegraph. Despite these efforts, Google remains the most popular general-purpose search engine with developers [46, 48], which is why we target it in our study.

5.2 Learning Barriers & Code Examples

Learning barriers in programming systems can arise from the environment and accompanying libraries [26]. Ko, et al. observed that learners sometimes knew what set of interfaces could achieve a behavior, but did not know how to use and coordinating them. The *use* and *coordination barriers* are hard to overcome without well-written documentation and code examples [26].

Prior studies point out that it is essential for API documentation to provide developers, especially those who are trying to learn a particular API, with sufficient and adequate code examples [33, 37, 43]. Parnin and Treude [33] observe that 90% of code-related blog posts contain code examples and snippets, with a media of three code snippets per post. McLellan, et al. [30] also summarize that the code examples supported several different learning activities, such as understanding the libraries, protocols, and usage context.

These studies echo our findings that 93.4% of the problems that learners encountered were successfully solved by finding relevant API and code implementations (43.9% & 49.5% respectively, Section 3.3), and the most successful searches are those that consult resources with examples, such as documentation and official/third-party tutorials.

5.3 End-User Programming and Spreadsheets

End-user programming is defined as "programming to achieve the result of a program primarily for personal, rather public use" [25]. Unlike professional developers who are employed to build, maintain and test software over time, end-user developers write programs to support goals in their own domains of expertise [12, 25, 31]. According to Scaffidi, et al., over 45 million end users reported that they "used spreadsheets or databases" in workplaces in 2001 [40]; however, only 20% of these workers indicated that they also "do programming." However, 44% of spreadsheets contained formulas, providing evidence of programming in spreadsheets [13].

Various studies have been done on spreadsheets, such as testing [38], code smells in formulas [12, 17, 22] and tools for spreadsheet formula transformation [16]. However, none study how end-users learn to program in spreadsheets, what challenges they encounter during programming, and how they approach solutions. Our work, while not specifically targeting end-users, provides some insights challenges encountered when learning VBA automation.

6 CONCLUSION AND FUTURE WORK

We studied how graduate students search when solving programming tasks in a new language. We find that subsequent programming language learners search with the purposes of exploring for example code, designing new features and understanding why code performs as it does. Learners composed more verbose queries, and required longer time to scan through the search result than professional developers did [39]. Consulting colleagues/friends in person could improve the chance to solve a problem. Consulting documentation and tutorials is more indicative of success in a search than consulting Q&A sites. Queries that focus on how something works are more likely to succeed when being composed with the structure, "*How to do...* ". Learns frequently borrowed terms from languages with which they are familiar when searching for examples in an unfamiliar language. The term borrowing queries were more likely to success than the typical query on average.

For future work, a replication study with a more diverse and larger set of subsequent programming language learners is suggested to better generalize how learners search. Techniques that can determine the relative APIs/libraries based on the context in the queries, provide query reformulation suggestions according to the search goals, map APIs between programming languages and present sample code are suggested to support more programming languages. Another suggestion for future work is to explore how to detect and correct the term mismatch in queries during the language migration.

ACKNOWLEDGMENTS

This material is based upon work supported by the National Science Foundation under Grant No. 1645136 and Grant No. 1749936.

REFERENCES

[1] William R. Aue, Amy Criss, and Matthew D. Novak. 2017. Evaluating mechanisms of proactive facilitation in cued recall. *Journal of Memory and Language* 94 (1 6 2017), 103–118.

[2] Gina R. Bai, Brian Clee, Nischal Shrestha, Carl Chapman, Cimone Wright, and Kathryn T. Stolee. 2019. Exploring Tools and Strategies Used During Regular Expression Composition Tasks. In *IEEE/ACM International Conference on Program Comprehension (ICPC 2019)*.

[3] Sushil Bajracharya, Trung Ngo, Erik Linstead, Yimeng Dou, Paul Rigor, Pierre Baldi, and Cristina Lopes. 2006. Sourcerer: A Search Engine for Open Source Code Supporting Structure-based Search. In *Symposium on Object-oriented Programming Systems, Languages, and Applications (OOPSLA '06)*. 681–682.

[4] Chetan Bansal, Thomas Zimmermann, Ahmed Hassan Awadallah, and Nachiappan Nagappan. 2019. The Usage of Web Search for Software Engineering. (2019). arXiv:cs.SE/1912.09519

[5] Veronika Bauer, Jonas Eckhardt, Benedikt Hauptmann, and Manuel Klimek. 2014. An Exploratory Study on Reuse at Google. In *International Workshop on Software Engineering Research and Industrial Practices (SER&IPs 2014)*. 14–23.

[6] Matt Bower and Annabelle McIver. 2011. Continual and Explicit Comparison to Promote Proactive Facilitation During Second Computer Language Learning. In *Conference on Innovation and Technology in Computer Science Education (ITiCSE '11)*. 218–222.

[7] Joel Brandt, Mira Dontcheva, Marcos Weskamp, and Scott R. Klemmer. 2010. Example-centric Programming: Integrating Web Search into the Development Environment. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '10)*. 513–522.

[8] 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 (CHI '09)*. 1589–1598.

[9] Chunyang Chen, Zhenchang Xing, and Yang Liu. 2019. What's Spain's Paris? Mining Analogical Libraries from Q&A Discussions. *Empirical Softw. Engg.* 24, 3 (June 2019), 1155–1194.

[10] Tonya Custis and Khalid Al-Kofahi. 2007. A New Approach for Evaluating Query Expansion: Query-document Term Mismatch. In *Proceedings of the International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '07)*. 575–582.

[11] Frederico A. Durão, Taciana A. Vanderlei, Eduardo S. Almeida, and Silvio R. de L. Meira. 2008. Applying a Semantic Layer in a Source Code Search Tool. In *Symposium on Applied Computing (SAC '08)*. 1151–1157.

[12] F. Hermans and B. Jansen and S. Roy and E. Aivaloglou and A. Swidan and D. Hoepelman. 2016. Spreadsheets are Code: An Overview of Software Engineering Approaches Applied to Spreadsheets. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Vol. 5. 56–65.

[13] Marc Fisher and Gregg Rothermel. 2005. The EUSES Spreadsheet Corpus: A Shared Resource for Supporting Experimentation with Spreadsheet Dependability Mechanisms. In *Workshop on End-user Software Engineering (WEUSE I)*. 1–5.

[14] Steve Fox, Kuldeep Karnawat, Mark Mydland, Susan Dumais, and Thomas White . 2005. Evaluating Implicit Measures to Improve Web Search. In *ACM:TOIS* (acm:tois ed.), Vol. 23. 147–168.

[15] Sonia Haiduc, Gabriele Bavota, Andrian Marcus, Rocco Oliveto, Andrea De Lucia, and Tim Menzies. 2013. Automatic query reformulations for text retrieval in software engineering. In *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, 842–851.

[16] Felienne Hermans and Danny Dig. 2014. BumbleBee: A Refactoring Environment for Spreadsheet Formulas. In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2014)*. 747–750.

[17] F. Hermans, M. Pinzger, and A. van Deursen. 2012. Detecting code smells in spreadsheet formulas. In *2012 28th IEEE International Conference on Software Maintenance (ICSM)*. 409–418.

[18] Reid Holmes and Gail C. Murphy. 2005. Using Structural Context to Recommend Source Code Examples. In *Proceedings of the International Conference on Software Engineering (ICSE '05)*. 117–125.

[19] J. Holvitie, T. Rajala, R. Haavisto, E. Kaila, M. Laakso, and T. Salakoski. 2012. Breaking the Programming Language Barrier: Using Program Visualizations to Transfer Programming Knowledge in One Programming Language to Another. In *2012 IEEE 12th International Conference on Advanced Learning Technologies*. 116–120.

[20] Jeff Huang and Efthimis N. Efthimiadis. 2009. Analyzing and Evaluating Query Reformulation Strategies in Web Search Logs. In *Proceedings of the ACM Conference on Information and Knowledge Management (CIKM '09)*. 77–86.

[21] M. Hucka and M.J. Graham. 2018. Software search is not a science, even among scientists: A survey of how scientists and engineers find software. *Journal of Systems and Software* 141 (2018), 171 – 191.

[22] B. Jansen and F. Hermans. 2015. Code smells in spreadsheet formulas revisited on an industrial dataset. In *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 372–380.

[23] Bernard J. Jansen, Amanda Spink, Chris Blakely, and Sherry Koshman. 2007. Defining a Session on Web Search Engines: Research Articles. *J. Am. Soc. Inf. Sci. Technol.* 58, 6 (April 2007), 862–871.

[24] Thorsten Joachims, Laura Granka, Bing Pan, Helene Hembrooke, Filip Radlinski, and Geri Gay. 2007. Evaluating the Accuracy of Implicit Feedback from Clicks and Query Reformulations in Web Search. *ACM Trans. Inf. Syst.* 25, 2, Article 7 (April 2007).

[25] Amy J. Ko, Robin Abraham, Laura Beckwith, Alan Blackwell, Margaret Burnett, Martin Erwig, Chris Scaffidi, Joseph Lawrance, Henry Lieberman, Brad Myers, Mary Beth Rosson, Gregg Rothermel, Mary Shaw, and Susan Wiedenbeck. 2011. The State of the Art in End-user Software Engineering. *ACM Comput. Surv.* 43, 3, Article 21 (April 2011), 44 pages.

[26] Amy J. Ko, Brad A. Myers, and Htet Htet Aung. 2004. Six Learning Barriers in End-User Programming Systems. In *Proceedings of the 2004 IEEE Symposium on Visual Languages - Human Centric Computing (VLHCC '04)*. IEEE Computer Society, Washington, DC, USA, 199–206.

[27] Amy J. Ko, Brad A. Myers, Michael J. Coblenz, and Htet Htet Aung. 2006. An Exploratory Study of How Developers Seek, Relate, and Collect Relevant Information During Software Maintenance Tasks. *IEEE Trans. Softw. Eng.* 32, 12 (Dec. 2006), 971–987.

[28] Otávio Augusto Lazzarini Lemos, Sushil Krishna Bajracharya, and Joel Ossher. 2007. CodeGenie: A Tool for Test-driven Source Code Search. In *Companion to the ACM SIGPLAN Conference on Object-oriented Programming Systems and Applications Companion (OOPSLA '07)*. 917–918.

[29] Philip Mayer and Alexander Bauer. 2015. An Empirical Analysis of the Utilization of Multiple Programming Languages in Open Source Projects. In *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering (EASE '15)*. ACM, New York, NY, USA, Article 4, 10 pages.

[30] S. G. McLellan, A. W. Roesler, J. T. Tempest, and C. I. Spinuzzi. 1998. Building More Usable APIs. *IEEE Software* 15, 3 (May 1998), 78–86.

[31] Bonnie A. Nardi. 1993. *A Small Matter of Programming: Perspectives on End User Computing*. MIT Press, Cambridge, MA, USA.

[32] H.James Nelson, Gretchen Irwin, and David E. Monarchi. 1997. Journeys up the mountain: Different paths to learning object-oriented programming. *Accounting, Management and Information Technologies* 7, 2 (1997), 53 – 85.

[33] Chris Parnin and Christoph Treude. 2011. Measuring API Documentation on the Web. In *Proceedings of the 2Nd International Workshop on Web 2.0 for Software Engineering (Web2SE '11)*. ACM, New York, NY, USA, 25–30.

[34] Md Masudur Rahman, Jed Barson, Sydney Paul, Joshua Kayan, Federico Andres Lois, Sebastian Fernandez Quezada, Christopher Parnin, Kathryn T. Stolee, and Baishakhi Ray. 2018. Evaluating how developers use general-purpose web-search for code retrieval. In *Proceedings of the 15th International Conference on Mining Software Repositories, MSR 2018, Gothenburg, Sweden, May 28-29, 2018*. 465–475.

[35] Mohammad M. Rahman, Chanchal K. Roy, and David Lo. 2016. RACK: Automatic API Recommendation Using Crowdsourced Knowledge. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, Vol. 1. 349–359.

[36] Mohammad M. Rahman, Chanchal K. Roy, and David Lo. 2019. Automatic query reformulation for code search using crowdsourced knowledge. *Empirical Software Engineering* (21 Jan 2019).

[37] M. P. Robillard. 2009. What Makes APIs Hard to Learn? Answers from Developers. *IEEE Software* 26, 6 (Nov 2009), 27–34.

[38] S. Roy, F. Hermans, and A. van Deursen. 2017. Spreadsheet testing in practice. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 338–348.

[39] Caitlin Sadowski, Kathryn T. Stolee, and Sebastian Elbaum. 2015. How Developers Search for Code: A Case Study. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2015)*. 191–201.

[40] Christopher Scaffidi, Mary Shaw, and Brad Myers. 2005. Estimating the Numbers of End Users and End User Programmers. In *Symposium on Visual Languages and Human-Centric Computing (VLHCC '05)*. IEEE Computer Society, Washington, DC, USA, 207–214.

[41] Jean Scholtz and Susan Wiedenbeck. 1990. Learning second and subsequent programming languages: A problem of transfer. *International Journal of Human-Computer Interaction* 2, 1 (1990), 51–72.

[42] Nischal Shrestha, Titus Barik, and Chris Parnin. 2018. It's Like Python But: Towards Supporting Transfer of Programming Language Knowledge. In *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*. IEEE, 177–185.

[43] Nischal Shrestha, Colton Botta, Titus Barik, and Chris Parnin. 2020. Here We Go Again: Why Is It Difficult for Developers to Learn Another Programming Language?. In *2020 IEEE/ACM 42nd International Conference on Software Engineering*. To appear.

[44] Susan Elliott Sim, Megha Agarwala, and Medha Umarji. 2013. *A Controlled Experiment on the Process Used by Developers During Internet-Scale Code Search*. New York, NY, 53–77.

[45] S. E. Sim, C. L. A. Clarke, and R. C. Holt. 1998. Archetypal source code searches: a survey of software developers and maintainers. In *Program Comprehension, 1998.*

IWPC '98. Proceedings., 6th International Workshop on. 180–187.

[46] Susan Elliott Sim, Medha Umarji, Sukanya Ratanotayanon, and Cristina V. Lopes. 2011. How Well Do Search Engines Support Code Retrieval on the Web? *ACM Trans. Softw. Eng. Methodol.* 21, 1, Article 4 (Dec. 2011), 25 pages.

[47] Singer, Janice and Lethbridge, Timothy and Vinson, Norman and Anquetil, Nicolas. 1997. An Examination of Software Engineering Work Practices. In *Proceedings of the 1997 Conference of the Centre for Advanced Studies on Collaborative Research (CASCON '97)*. IBM Press, 21–36.

[48] Kathryn T. Stolee, Sebastian Elbaum, and Daniel Dobos. 2014. Solving the Search for Source Code. *ACM Trans. Softw. Eng. Methodol.* 23, 3, Article 26 (June 2014), 45 pages.

[49] Jeffrey Stylos and Brad Myers. 2006. Mica: A Web-Search Tool for Finding API Components and Examples. 195–202.

[50] Neeraja Subrahmaniyan, Laura Beckwith, Valentina Grigoreanu, Margaret Burnett, Susan Wiedenbeck, Vaishnavi Narayanan, Karin Bucht, Russell Drummond, and Xiaoli Fern. 2008. Testing vs. Code Inspection vs. What Else? Male and Female End Users' Debugging Strategies. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI'08)*. Association for Computing Machinery, New York, NY, USA, 617–626.

[51] Federico Tomassetti and Marco Torchiano. 2014. An Empirical Assessment of Polyglot-ism in GitHub. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering (EASE '14)*. Article 17, 4 pages.

[52] Meng-Jung Tsai, Jyh-Chong Liang, Huei-Tse Hou, and Chin-Chung Tsai. 2012. University students' online information searching strategies in different search contexts. *Australasian Journal of Educational Technology* 28, 5 (Jul. 2012).

[53] Medha Umarji, Susan Sim, and Crista Lopes. 2008. Archetypal Internet-Scale Source Code Searching. *International Federation for Information Processing Digital Library; Open Source Development, Communities and Quality;* 7, 257–263.

[54] 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 Softw. Engg.* 22, 6 (Dec. 2017), 3149–3185.

[55] Jun Xu, Wei Wu, Hang Li, and Gu Xu. 2011. A Kernel Approach to Addressing Term Mismatch. In *Proceedings of the International Conference Companion on World Wide Web (WWW '11)*. 153–154.

[56] Hongyu Zhang, Anuj Jain, Gaurav Khandelwal, Chandrashekhar Kaushik, Scott Ge, and Wenxiang Hu. 2016. Bing Developer Assistant: Improving Developer Productivity by Recommending Sample Code. In *Proceedings of the International Symposium on Foundations of Software Engineering (FSE 2016)*. 956–961.

[57] Xiangmin Zhang, H.G.B. Anghelescu, and Xiao-Jun Yuan. 2005. Domain knowledge, search behaviour, and search effectiveness of engineering and science students: An exploratory study. *Information Research* 10 (01 2005).

[58] Le Zhao and Jamie Callan. 2012. Automatic Term Mismatch Diagnosis for Selective Query Expansion. In *Proceedings of the International Conference on Research and Development in Information Retrieval (SIGIR '12)*. 515–524.