Evaluating How Developers Use General-Purpose Web-Search for Code Retrieval

Md Masudur Rahman
University of Virginia
masud@virginia.edu

Jed Barson
University of Virginia
jb3bt@virginia.edu

Sydney Paul
Clemson University
sepaul@g.clemson.edu

Joshua Kayani
North Carolina State University
jkayani@ncsu.edu

Federico Andrés Lois
Codealike, Argentina
Federico.lois@corvalius.com

Sebastián Fernandez Quezada
Codealike, Argentina
sebastian.quezada@corvalius.com

Christopher Parnin
North Carolina State University
cjparnin@ncsu.edu

Kathryn T. Stolee
North Carolina State University
ktstolee@ncsu.edu

Baishakhi Ray
University of Virginia
rayb@virginia.edu

ABSTRACT

Search is an integral part of a software development process. Developers often use search engines to look for information during development, including reusable code snippets, API understanding, and reference examples. Developers tend to prefer general-purpose search engines like Google, which are often not optimized for code related documents and use search strategies and ranking techniques that are more optimized for generic, non-code related information.

In this paper, we explore whether a general purpose search engine like Google is an optimal choice for code-related searches. In particular, we investigate whether the performance of searching with Google varies for code vs. non-code related searches. To analyze this, we collect search logs from 310 developers that contains nearly 150,000 search queries from Google and the associated result clicks. To differentiate between code-related searches and non-code-related searches, we build a model which identifies the code intent of queries. Leveraging this model, we build an automatic classifier that detects a code and non-code related query. We confirm the effectiveness of the classifier on manually annotated queries where the classifier achieves a *precision* of 87%, a *recall* of 86%, and an *F1-score* of 87%. We apply this classifier to automatically annotate all the queries in the dataset. Analyzing this dataset, we observe that code related searching often requires more effort (e.g., time, result clicks, and query modifications) than general non-code search, which indicates code search performance with a general search engine is less effective.

ACM Reference format:
Md Masudur Rahman, Jed Barson, Sydney Paul, Joshua Kayani, Federico Andrés Lois, Sebastián 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 MSR '18: 15th International Conference on Mining Software Repositories , Gothenburg, Sweden, May 28–29, 2018 (MSR '18),* 11 pages.
https://doi.org/10.1145/3196398.3196425

MSR '18, May 28–29, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5716-6/18/05…$15.00
https://doi.org/10.1145/3196398.3196425

1 INTRODUCTION

Search plays an important role in fulfilling users' information needs. In particular, search for code has been an integral part of software development processes in the past [3, 28, 38, 39]. Developers often use a search engine for various information needs, including finding reusable code snippets, understanding APIs, locating reference examples, learning unfamiliar concepts, remembering syntactic details, identifying appropriate third-party libraries, and debugging [16, 28, 39]. In the literature, code search has been studied extensively and researchers proposed many approaches to improve code search performance [1, 10, 13, 15, 18–20, 22–24, 26, 27, 32, 37, 41].

In practice, to support the increasing need for code search in software development, several commercial search engines have been developed, such as Google Code Search [11], Black Duck Open Hub Code Search [33], and others (e.g., [17, 29]). Unfortunately, many of them (e.g., [11, 33]) are now obsolete. Thus, programmers tend to turn to a general purpose search engine (e.g., Google [12], Yahoo [40], Bing [4]) to search for code [16, 31, 36], and software [16], and they rarely use a dedicated code search engines [16]. Among several general-purpose search engines, *Google* has been found to be the most frequently used search engine for software development related searches [31].

These general purpose search engines (GPSE) are usually optimized for textual search [16] and treat code as plain text when used for searching code. Thus, they tend to ignore the underlying semantics of the code. In fact, Google's dedicated code search engine [9] used an additional layer of n-gram based regular expression matching technique to cater the special needs of code search. Using GPSE for code search might be a reason that despite the tremendous increase in online resources (e.g., GitHub, SourceForge, StackOverflow, API documentation), suitably locating reusable source code still remains a major challenge—developers often fail to locate the intended code using different search approaches including web-search [16]. Surveys have shown that developers look at an average

of 3.5 snippets of code before finding something useful for their task at hand [36].

Despite their limitations, GPSEs are the most popular choice for code search and will continue to be like that, in all likelihood, because they are lightweight, easy to use, and have sophisticated web interfaces [22]. Thus, it is worthwhile to evaluate *how* GPSE*s perform while used for code vs. general non-code related search* so that we can better understand GPSE's shortcomings in code domain and tune them accordingly. In this paper, we shed some empirical light to explore this question by studying Google search [12]. In particular, we investigate how the search behavior of users and performance of the search engine vary for code related searches compared to non-code related searches.

To this end, we analyzed 14 months of web search logs from 310 developers using a Google Chrome plugin, which tracks browsing activities of its users [6]. In total, we analyzed 149, 610 Google search queries. Since these are web search logs of the developers during their working time, the logs contain both code and non-code related queries, although they are not annotated as such. First, we develop an automated technique to classify these queries to code vs. non-code. We leverage Stack Overflow [34] tags to extract code-related tokens. A *codeness* score is calculated for each query based on how many stack-overflow tokens are present in it. A higher codeness score indicates the query is more likely to be code related. A manual evaluation shows that the classifier achieves a *precision* of 87%, a *recall* of 86% to successfully classify code vs. non-code queries. Using this classifier we find 88,577 (59.21%) code related queries and 61,033 (40.79%) non-code queries in our dataset. We use this annotated data to analyze the differences between code and non-code related search for GPSE. We study both query characteristics (RQ1) and developers' effort (RQ2 & RQ3) and find that:

(1) A single code query is, in general, larger and uses a smaller vocabulary than a non-code query (see RQ1).
(2) To retrieve the intended answer, users have to spend more time on a single code query and have to modify the queries more often than the non-code queries (see RQ 2).
(3) To complete a code related search task, users require more queries, more URLs clicked, and overall more time than non-code related search tasks (see RQ 3).

Several empirical studies have been performed to identify how developers search for code [28], what type of code issues they search on Web [39], and how the performance varies when developers search for code with different search engines [31]. Yet, it remains less explored how code search is different than searching for general information, i.e. non-code search. Little is known about how the GPSE performs on code search compared to others. In this paper, we seek to answer these questions. In summary, we make the following contributions:

- Build and evaluate a novel technique to automatically classify a search query to code vs. non-code (Section 4.2).
- Analyze the query characteristics and how it differs between code vs. non-code queries in general-purpose web search (RQ 1).
- Analyze users' effort in retrieving the intended result for code and non-code related queries (RQ 2 and RQ 3).

We organize the rest of the paper as follows. We start by describing background information and research questions in Section 2. Then we discuss our code intent model in Section 3. We discuss our methodology in details including data collection, query extraction and annotation, and classifier evaluation in Section 4. After that, we analyze our experimental results in Section 5. We discuss the implication of our code intent model and findings in Section 6. Then we discuss related work in Section 7, possible threats to validity in Section 8, and conclude in Section 9.

2 RESEARCH QUESTIONS

Typically, code related artifacts (e.g., source code, bug reports, API documentation, etc.) are different from general documents, such as news, Wikipedia articles, or other non-code information sources [14]. While the latter is primarily composed of natural languages, the code related documents can be a mix of programming and natural languages. However, GPSE treats source code as text and ignores all the programming language related features. For example, the source code is *less ambiguous* than natural language so that the code can be interpreted by a compiler. However, GPSE ignores the syntactic and semantic features of the source code and thus, cannot interpret the underlying behavior. Thus, using GPSE, locating similar code or retrieving code examples becomes difficult unless both query and the documents use similar vocabulary [36]. But, since source code contains *open vocabulary* (i.e. developers can coin new variable names without changing the semantics of the programs), searching for code somewhat becomes a guessing game for GPSE.

In this paper, we empirically evaluate the impact of using GPSE for code related search. In particular, we investigate how code search differs from non-code related search with GPSE in two dimensions: (i) query characteristics (RQ1), and (ii) users' effort (RQ2 and RQ3). To this end, we explore the following research questions:

> **RQ1.** How do query characteristics differ for code and non-code queries?

To explore this RQ, we analyze how linguistically the two queries are different. In particular, we check whether query length varies for code and non-code search. We also study the vocabulary sizes and vocabulary choices between the two.

> **RQ2.** How do search behaviors vary for code and non-code related queries?

In this RQ, we explore different search behaviors of users, including how much time they spend on search results, how many websites they visit, and how often they modify their search queries. We also analyze how this behavior varies for code and non-code related searches.

> **RQ3.** How do task sessions vary for code and non-code related search tasks?

Often, several queries can be related to same web search task. To explore this RQ, we identify sequences of related queries as *task sessions* (Section 4.1). Next, we analyze how many queries, how much time, and how many website visits users require to complete a task. We also analyze how these task level interactions differ for code related search compared to non-code.

3 CODE INTENT ANALYSIS

We assume that if a query contains more code related tokens (e.g., "javascript", "C#", "json", "visual-studio"), it indicates more code intent. To automatically estimate such intent, we build an analysis technique that assigns a code intent score to each query. We call this score as *codeness score*. In this section, we discuss our analysis technique in details.

3.1 Code Intent of Tokens

To construct the model, we first collect a list of code related token set (S). We leverage StackOverflow (SO) [34] (May 2017 data dump) which is an online Q/A forum where developers often discuss their programming related issues. A post in SO can be associated with tag(s), which are given manually. However, not all the tags are equally strong indicators of code intent. We deal with such scenarios as follows:

Firstly, we filter out ambiguous tags from our token set. SO tags often co-occur with other tags and thus there might exist some tags which always co-occur and never occur alone in any post. These tags might not be an indication of code token. For example, "unbox" tag never occurs alone but occurs with "haskell" tag [35] which is a code token. Such tag (i.e. "unbox") might not be an indicator of code intent. To remove such unwanted noise, we filter out all the tags which never occur alone in any post. Additionally, we remove all the post with multiple tags. Thus the frequency of a tag is the count of its single occurred posts only. This process reduced the number of selected tags drastically from $46.3K$ to $19.8K$

Secondly, we assign a *codeness score* for each tag in our filtered code token set (S). We assume that the popularity of a tag on SO is the indicator of its code intent. Higher frequency (i.e. popularity) indicates strong code intent. However, the raw frequency might lead to incorrect code intent estimation. For example, in Table 1, the frequency (i.e. count) difference between "android" and "java" shows "android" carries much higher code intent than "java" which is not completely accurate estimation. To mitigate such frequency difference bias, we use sub-linear scaling. If a tag x occurs n times then the *codeness score*, $f(x)$, of that token is given by equation 1,

$$f(x) = \begin{cases} 1 + log2(n), & \text{if } x \in S \\ 0, & \text{if } x \notin S \end{cases} \quad (1)$$

where S is the code token set. Note that, if a token is not in the code token set (S) its *codeness score* is 0 and that token is considered as a non-code token. n is the frequency of token x across all Stack Overflow posts.

Now, considering previous "android" vs "java" example, we see the *codeness score* are 17.55 and 17.13 (in Table 1) which shows both tag are of similar code strength. In contrast, in Table 1 *codeness score* of "lucene" is 10.18, which indicates its code intent is less compared to "android" or "java". Some code tokens in different count ranges, and their *codeness score* can be found in Table 1.

3.2 Code Intent of Queries

We leverage the token level codeness score to compute the *codeness score* of the query. The *codeness score* (*cscore*) of a query is calculated by summing up the code score of its tokens as in equation 2

Table 1: Sample code tokens' count (single occurrence) and their *codeness score*

Tags	count	cscore	Tags	count	cscore
android	96210	17.55	css3	982	10.94
java	71869	17.13	applescript	956	10.9
php	71390	17.12	lucene	579	10.18
javascript	70248	17.1	coffeescript	579	10.18
python	53993	16.72	firefox-addon	268	9.07
jquery	52705	16.69	livecode	268	9.07
c#	48898	16.58	jasmine	86	7.43
mysql	41684	16.35	codeigniter-3	86	7.43
c++	41283	16.33	miniprofiler	4	3
r	30176	15.88	idocscript	1	1

Table 2: Sample query and their codeness score assigned by our model

	Query	Code Score	Type
1	javascript mp3 play time	40.71	Code
2	javascript get track length from meta data	48.57	Code
3	how to perform xml serialization for parameterless constructor in c#	67.33	Code
4	elasticsearch.net & nest installed post nuget source control stop notification	49.36	Code
5	acer e700 review	7.07	Noncode
6	houston luxury suv rental	0.00	Noncode
7	messi curly goal	2.58	Noncode

$$cscore(Q) = \sum_{i=1}^{m} f(x_i) \quad (2)$$

where x_i is the i^{th} token of the query Q of length m and $f(x_i)$ is the *codeness score* of token x_i as in equation 1.

If the *codeness score* of a query is high it is considered to have a high code intent. In this way, the model assigns a code intent to each query. Some sample queries with their *codeness score* is shown in Table 2.

4 METHODOLOGY

In this section, we start with explaining our data collection and query extraction approach in detail. Then we present both manual and automatic query annotation process. After that, we describe search tasks extraction and classification method.

4.1 Study Subject

Our search log data was collected from developers who installed a proprietary Google Chrome Web Tracking plugin [6]. The plugin tracks all the web browsing activities which are processed and analyzed to understand how developer work and learn. Thus in our dataset most of the users are developers either acting as team leaders or performing technical tasks and the activity includes search query and clicked web page visit information.

The data collection period spanned 14 months starting from December 2014 to January 2016. There are a total of 149, 610 queries of 310 users (See Table 3).

The dataset contains information about *activity sessions* for each user, which represent a user's active development time [8]. Each user can have many activity sessions. Each activity session contains events in the web browser, such as search queries and results clicked, as well as non-browser events, such as IDE interactions. For the browser events, the logs provide information on the clicked results, specifically, the URL and page title. All events have a start time and an end time.

These activity sessions provide a useful boundary of continuous activity for a user, but finer granularity is needed as the logs contain information about browser interactions as well as non-browser interactions. Further, we want a notion of related web activities, since users often initiate consecutive yet unrelated search queries. After identifying consecutive related queries, we split the activity sessions into *task sessions*. This is accomplished by first identifying *edited queries*.

Identifying Edited Query: Users often modify their search query to give more specific information to the search engine. These query reformulations can expand the query by adding more terms or reduce the query by removing terms. If a query contains *at least one common term* with its previous query, and the queries come from the same activity session, we consider both queries as edited queries.

Composing Task Sessions: Task sessions capture continuous, related web browser interactions. We consider all browser events after one search query and before the next query as the *result exploration* activity for the former query; the web URLs of those activities are considered *clicked URL*.

To identify task sessions, first we explore all continuous sequences of edited queries and their associated results exploration activities. Each sequence of edited queries represents a task session. The remaining queries are all non-edited. Each non-edited query, along with its results exploration activities, forms its own task session.

Computing Search Query Time: Users spend time on the search page and on the web pages they click. The time between when a query is issued and when the next query is issued, or the activity session ends (whichever comes first), is referred to as the *query search time*. In the event that a user does not click any results, the *query search time* is computed as the time spent on the result page.

4.2 Query Classification

A query which is intended to solve any software development related issue is considered as a *code* query. For example, *reference code example* (e.g., "write in file java", and "how to get all textbox names inside table layout panel c#"), debugging (e.g., "asp.net mvc error page"), API usage, technical knowledge (e.g., "npm update all dependencies", "git bash mingw", and "qualities of good programmer") and other development related tasks are considered as code related query. A query which is not intended to solve any software development or programming task is considered as a *non-code* or general query. For example, "make your own comics", "review Galaxy Note Edge", and "d5300 amazon" are considered as non-code. To set the threshold and evaluate our classifier, we manually annotate queries to code and non-code.

Manual Query Annotation. From our dataset, we randomly sample 380 queries across users. Two researchers separately annotated those queries and resolved the disagreement by discussion. We measure Cohen's kappa coefficient (κ) [7] to find inter-annotator agreement, where a κ value of 1 indicates a complete agreement and a value of 0 indicates a complete disagreement. In our annotation we find a κ value of 0.85.

Evaluation Metrics. We use following metrics to evaluate our classifier:

Precision (P) - is the fraction of correct prediction of total query. Thus, $P = \frac{r}{d}$, where r is the number of correct prediction and d is the total query.

Recall (R) - is the fraction of correct prediction of the total ground truth. If t be the total ground truth, the recall is $R = \frac{r}{t}$.

F_1 *Score (F-1)* - is a single combined metric that trades off precision vs. recall by computing the harmonic mean of the two: $F_1 = 2 * \frac{precision * recall}{precision + recall}$.

Accuracy Evaluation. Figure 1 shows *precision*, *recall*, and *F1-score* with respect to **code query** in different *codeness score* thresholds. As the threshold increases, *precision* also increases. In contrast, *recall* decreases with the increase in threshold. However, *F1 Score* remains in between *precision* and *recall* in different thresholds. For a better comparison, it is important to maintain a balance between code and non-code query classification. Thus, we choose the threshold = 10 where the classifier achieve a better trade-off of $Precision = 87\%$, $Recall = 86\%$, and $F1Score = 87\%$.

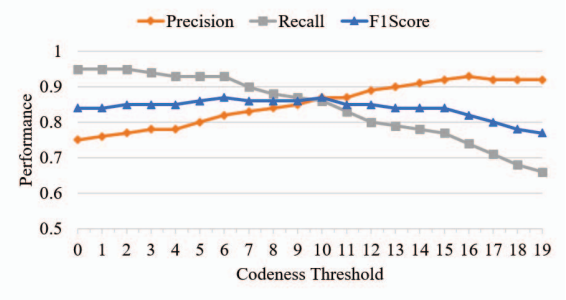

Figure 1: Classifier Evaluation

These results indicate that our model assigns *codeness score* which is effective in detecting query intent. In addition to separating code from non-code, the model also identifies the code specificity of a code query. A larger score indicates a strong code intent. Thus, we can further separate code related queries into different clusters based on different ranges of *codeness score*.

For our analysis, we need to classify all queries to code or non-code. Empirically, we set *codeness score* threshold to 10, which gives us a better trade-off for precision and recall. Details dataset statistics after the query classification can be found in Table 3.

Table 3: Dataset Statistics (Codeness Score Threshold = 10)

Query	#	%	User	Query/ User	User-Query Stats Min	Q1	Q2	Q3	Max
Code	88577	59.21	300	295.26	1	22	136.5	387.25	2593
Noncode	61033	40.79	296	206.19	1	15	85.5	294	1642
All	149610	100	310	482.61	1	28.75	207	676	3632

4.3 Extract and Classify Search Task

We analyze how user interaction varies for code and non-code related search tasks. First, we extract task information from the search log data. We define a search task as a set of the consecutive edited queries. We start with a query and consider all the subsequent queries which were edited from the previous query and stop when encountering a totally new query. Thus we extract all such tasks for all users. Applying this process, we extract a total of 108,313 tasks. Secondly, to analyze code and non-code task properties, we compute the *codeness score* for each task. We assign a representative query whose *codeness score* is maximum among all other queries in a task. We consider this maximum *codeness score* as the code intent for that task. Thus we assign a *codeness score* for all the tasks. Similar to query classification, we consider a task with a *codeness score* greater than a particular threshold (10 in our experiment) as code related task and non-code otherwise. Note that, in a task, a user might start with a lower code intent query and can add code token(s) gradually to increase the code intent of the whole task. Sample task session from our dataset can be found in Table 4.

4.4 Codeness Difference Calculation

Search engines (i.e. Google) often suggest query edits with the search results. This helps users to come up with their desired query for their informational need. To this end, we analyze what happens w.r.t. *codeness score* when users edit a code related query. If a query q is reformulated to q_r then we calculate their *codeness score* difference, $\Delta Codeness$, as in Equation 3.

$$\Delta Codeness = Codeness(q_r) - Codeness(q) \quad (3)$$

Here, a positive value of $\Delta Codeness$ indicates an increase, a negative value indicates a decrease and zero (0) value indicates no-change in code intent after reformulation. We compute the $\Delta Codeness$ for all the edited code related queries in three different settings: edited 1) only by adding term, 2) only by deleting term, and 3) overall, adding or/and deleting term.

5 RESULTS

In this section we discuss our experiments and results analysis of RQs.

RQ1. How do query characteristics differ for code and non-code queries?

To analyze query characteristics we filter out the duplicates to mitigate unwanted bias from query duplication. In this search log, we found 20.36% duplicate queries with duplication for both code (20%) and non-code (21%) queries.

1) How do code-related queries differ in length from non-code-related queries?

We begin with exploring the query length. Figure 2(a) shows that code related query length (i.e. number of tokens or words in a query) is often higher than the non-code, with statistical significance (confirmed by Wilcox statistical significance test with *medium* Cohen'D effect size). The average length (4.7) of the code related queries is higher than non-code (2.3). This implies that users tend to use more words to express a code related issue which is almost twice that needed to for a general non-code issue.

To dig into this further, we analyze how the query length varies with the increase of the code intent of the query (i.e. *codeness score*). Figure 2(b) shows the comparison in query length in different *codeness score* ranges. We see that often, queries with higher *codeness score* are longer in length. Note that by definition *codeness score* increases with the increase of code related tokens in a query. However, adding a non-code token does not an increase in *codeness score*. Thus, sharp increase of query length with *codeness score* in Figure 2(b) confirms that code related query are indeed more verbose.

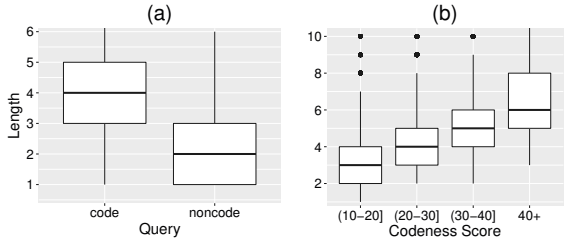

Figure 2: Query Length (# of words)

2) How do vocabulary varies for code and non-code query? For this analysis, we remove English stop-words (adopted from [21]) from the queries . We find that vocabulary of code is $28K$ which is much smaller than non-code query $45K$ though, in our annotated data, the number of code query (i.e. 59.21%) is higher than non-code (i.e. 40.79%). However, we observe that 43.48% of code vocabulary are common with non-code which is depicted in Figure 3. In Table 5 we list top frequently occur code, non-code, and common tokens with their frequency.

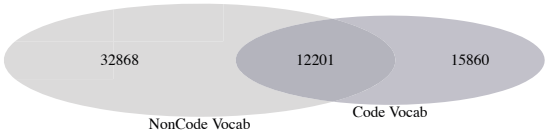

Figure 3: Vocabulary words statistics for code and non-code query

Code related queries are often intended for a particular programming language. To reduce the search space for relevant documents, it is important to understand what programming language the user intended to use. To explore this, we analyze how frequently user explicitly mentions the language by name in the query. We use a list of 100 popular programming languages [25] and search for these keywords in code related queries. We find that users mention language name in the code query 20% of the time and skip 80% of

Table 4: Sample Task Sessions (from Dataset)

Task	Edit Seq.	Query	Added Terms	Deleted Terms
Code	1	how to get mp3 playtime in c# from stream		
	2	javascript mp3 play time	javascript, play, time	how, to, c#, from, stream, playtime
	3	how to get mp3 play time length	how, to , get, length	javascript
	4	javascript function to get mp3 play length	javascript, function	how , time
	5	javascript read mp3 metadata	read, metadata	function, to, get, play, length
Noncode	1	enterprise luxury suv		
	2	luxury suv rentals houston	rentals, houston	enterprise

Table 5: Top query words and frequency for Code, Noncode and common words between them (w/o English Stopword)

Code		Common		NonCode	
Token	Freq	Token	Total Freq	Token	Freq
c#	6165	string	1179	2015	262
sql	2604	add	982	de	204
windows	2587	type	950	define	153
javascript	1966	error	855	meme	108
server	1936	create	847	uk	104
jquery	1713	change	844	dell	99
studio	1696	list	826	la	94
visual	1639	set	809	day	83
file	1443	object	782	price	82
string	1160	array	741	world	79
mvc	1144	table	695	south	79
web	1038	2015	687	movie	79
code	979	date	656	lyrics	76
add	946	find	645	weather	75
type	929	time	633	top	73
asp.net	915	check	627	road	73

Table 6: Most Frequently Mentioned PLs. 20.10% **of code queries mention at least one PL name**

Top	Language	Freq.	Top	Language	Freq.
1	c#	6274	11	python	192
2	sql	2586	12	c	145
3	javascript	1970	13	bash	131
4	.net	683	14	go	130
5	php	469	15	ruby	102
6	powershell	388	16	crystal	94
7	assembly	267	17	r	87
8	c++	255	18	logo	58
9	java	250	19	s	56
10	icon	203	20	f#	53

the time. This indicates that most of the time developers do not mention language explicitly thus it is up to the search engine to guess which programming language the developers intended. The top mentioned programming languages with their query frequency is shown in Table 6.

> **Result 1:** *Code queries are linguistically different than non-code queries—they are longer and contain less vocabulary.*

RQ2. How do search behaviors vary for code and non-code related queries?

We investigate this question in three dimensions: (i) how much time users have to spend per query, (ii) how many websites they visit per query, and (iii) how many times users have to edit a query to retrieve the intended document. We will discuss them one by one.

1) How long do users spend searching for code-related issues compared to non-code-related issues?

We compare time duration for code and non-code query in Figure 4 (a). We see that in code related query users take slightly more time compared to non-code query search with the median time of 1min 20 sec and 1min 4 sec for code and for non-code queries respectively. Although this difference is statistically significant (Wilcox's Test), CohenD's effect size is negligible.

We further check whether time duration varies with *codeness score*. Figure 4 (b) shows that as the *codeness score* of queries increases users tend to spend slightly more time on searching (with negligible effect size). Thus, in reality, we do not see any major difference code and non-code queries w.r.t. the time users spend interacting with the search engine.

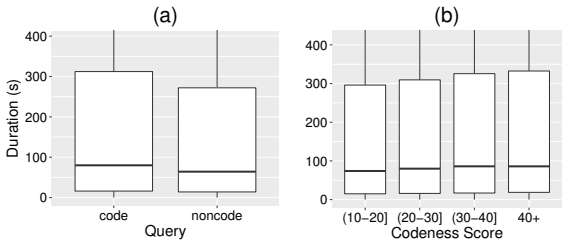

Figure 4: Query Browsing Duration

2) How many websites do people traverse when searching for code related issues compared to non-code related issues?

From Figure 5 (a), we see that there is no significant difference (confirmed by Wilcox statistical significance test and Cohen'D effect size) between code and non-code with a median of 4 for both query types. The average number of clicks per query is 11.4 for code and 12.8 for non-code. This result indicates no matter what types of problem users search for, they often visit a similar number of websites. One possible explanation is - after a certain number of clicks, users stop exploring results no matter whether the returned results are satisfactory or not. This hypothesis can be explained further by the results in Figure 5 (b). We see that there is no visible trend of the number of website visits in the different range of *codeness score*. We can conclude that users visit a similar number of website in general for all queries. This behavior can be considered as a common search behavior.

Furthermore, we observe that 22% of non-code queries require no website visits, which is higher when compared to 17.9% for code related search. This indicates that for non-code general search, users get relevant results from the search results page only (i.e. fact searching, summarized results from Google) or quickly realize

whether the resulting information is relevant at all. On the other hand, for code related search users need to click and see the content (most of the cases) of the website to judge whether the page contains relevant information or not.

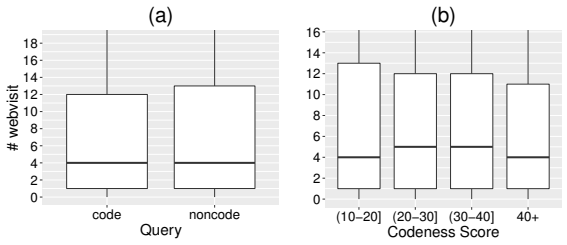

Figure 5: Web Visit Analysis for Query

3) How often do people have to modify their search when searching for code related issues compared to non-code related issues?

Depending on the the context, the user might add/insert terms to the query, delete terms, or both in order to reformulate the query. Such queries are called edited queries. The user keeps editing a query repeatedly until they are satisfied with the returned results. Thus, an ideal search engine would return the exact satisfactory documents when user issues a query for the first time. The more a query is edited, the more effort is needed from the user to find out relevant documents. To this end, we observe how users modify queries during searching.

We observe that in total 27.6% of queries are edited queries. In particular, 34.9% of the code queries are edited queries, which is significantly higher than 17.01% of non-code edited queries. This result indicates that search engine (i.e. Google) found it twice as difficult to understand code search compared to non-code search.

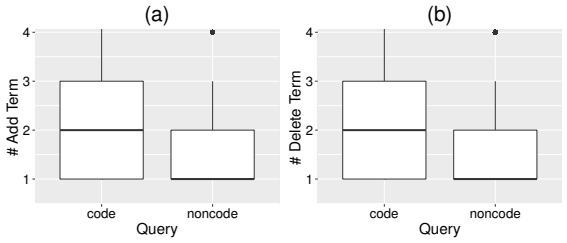

Figure 6: Add and Delete Term in Query Statistics for Code vs Non-code

Type of Query Edit. To explore further, we observe the type of edits users make during query modification. In Figure 6 we see that for non-code search users often add or delete one term at a time to generate their edited query. On the other hand, with code-related search, most of the time users often achieve an edited query by adding or deleting two terms.

This phenomenon can be observed in Table 7 which shows top unigram and bigram tokens added, as well as deleted terms found in our dataset for **code queries**. For instance, users often add "visual studio" to their query to clarify their search intent as something

Table 7: Term Statistics for Edited Code Query

Top Added Terms		Top Deleted Terms	
Unigram	Bigram	Unigram	Bigram
to	visual studio	to	visual studio
c#	c# winforms	in	when only
in	framework entity	for	success is
the	not working	and	status code
of	windows 8	with	returned success
windows	how to	c#	phone css
sql	model object	on	parameters in
a	unit test	object	not working
not	parameters in	from	is when
for	using c#	is	follow up
javascript	windows phone	the	code returned
how	when only	not	a status
server	what is	css	sky in
from	to add	a	object 2007
jquery	status code	javascript	model object

related to "visual studio" platform. Similarly, users often mention "using c#" term to indicate they want the solution in "c#" language. A similar conclusion can be drawn for the deleted term as well.

Edit vs Codeness Score. Figure 7 shows how *codeness score* changes when uses reformulate code related queries. We see that the developer achieve an edit by only adding terms, it often increases (i.e. positive median of *onlyAdd* in Fig. 7) code intent of the query. Not surprisingly, when developers modify a query by only deleting terms, *codeness score* decreases (i.e. negative median of *onlyDelete* in Fig. 7). However, we see that the median of *overallEdit* is 2 (i.e. positive). This indicates, when the developer reformulates (i.e. adding or/and deleting terms) a query, it often increases the code intent of the current query.

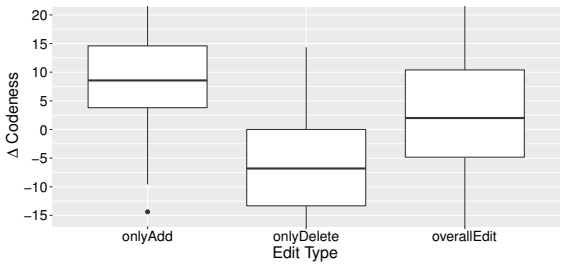

Figure 7: Query Edit Type vs Codeness

These results indicate that though in terms of time spending per query and website visit, there is no significant differences, users have to edit code queries more often than non-code queries. Thus, overall more effort is needed to query code using GPSE.

> **Result 2:** *Users modify code queries more often than non-code queries to retrieve desired results.*

RQ3. How do task sessions vary for code and non-code related search tasks?

Often users need multiple queries to complete a task. Thus, we further check how much effort is needed to compare a whole code vs. non-code task. We use the annotated tasks data and perform the following experiments.

Table 8: Number of Queries per Task

# Query	% Code	% Noncode
1	70.96	**90.76**
2	**16.41**	6.49
3	**6.22**	1.55
4	**2.94**	0.67
5	**1.51**	0.24
5+	**1.96**	0.29
Total	100%	100%

1) How many queries do users need to complete a search task?

In Table 8, we see that the number of single query non-code task is 90.76% which is significantly higher compared to 70.96% for code task. On the other hand, the percentage of task consist of two query is 16.41% for code which is significantly higher than 6.49% for the non-code. As the number of queries per task increases (2, 3, 4, 5, 5+) (Table 8) the percentage of the task is getting higher for code task compared to a non-code task. Overall, code task requires significantly (Wilcox significance test with *small* Cohen'D effect size) higher number of queries than non-code. On the other hand, the number of queries to complete a task can be considered as the amount of effort and interaction required from users. This implies that the effort required to complete a search task is higher for code related search compared to non-code search.

In addition, we see that (in Table 8) for code related search almost 2% of the time user made more than 5 edits to the queries which is around 85% higher compared to non-code search (i.e. 0.29%). This result indicates that users remain patient with the search engine when they look for the code. This also indicates a code search task is more complex which required more edits on queries to properly convey the information need to the search engine compared to non-code general search.

2) How much time do users need to complete a search task?

In this RQ we analyze how much time users spend on the search task. We sum up all the queries' activity time in a task to get the total duration of a search task. From Figure 8 (a), we see that most of the code tasks required more time compared to non-code search tasks. The median time to complete a code search task is around 2 min 53 sec and the median time to complete a non-code search task is 1 min 35 sec. We see that generally, users spend almost twice the time for code related search compared to non-code. This result confirms the finding of RQ 3-1 that users are more patient for code related search than for non-code related search.

3) How many different website visits are required to complete a search task?

Similar to time duration, we also analyze the number of web visits required for different tasks. In Figure 8 (b), we see that most of the code search tasks required more web visits than non-code search tasks. In RQ 2, we find no specific pattern in the number of web visits between code and non-code related query. However, here we see that the median of the number of web visits is 8 for code task, which is higher than the median of the number of web visits for non-code task (6). The increasing number of queries in code search tasks (as in RQ 3-1) might contribute to the difference in number of web visits for the search task.

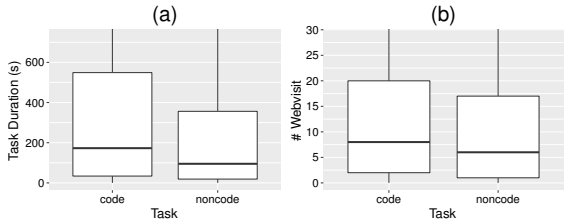

Figure 8: Task Duration and Webvisit Statistics for Code vs Non-code

Result 3: *Users spend significantly more effort for code related task than non-code related task in terms of number of queries, task completion time, and number of website visit.*

Discussion

In summary, we find that code and non-code queries have different query characteristics. Also, user needs to put more effort to retrieve the intended results for code than non-code with a GPSE.

While our work primarily analyze code vs. non-code queries, there could be further refinement possible for different kinds of programming tasks and information needs behind the search. For example, analyzing developers search queries on web, Xia et al. [39] identified search tasks into seven different categories. We also see similar pattern while manually annotating 178 code queries. Some sample queries with their task categories are shown in Table 9.

Table 9: Sample code queries with their task categories (from Dataset)

	Task Type (short name)	Query Example
1	General Search (gen)	c# property naming guidelines
2	Debugging (debug)	jira not loading images, Attempt to load Oracle client libraries threw BadImageFormatException This problem will occur ...
3	Programming (prog)	how to call class function in webservice c#
4	Code Reuse (reuse)	GWTP template maven
5	Tools (tool)	php online debugger, lighttable
6	Database (db)	sqlserver database rename
7	Testing (test)	get protected member unit testing

Figure 9 further shows how query length, query browsing duration, and website visit can vary for different code tasks. We see that most of the *debug* queries' length are higher compared to others and general code queries (i.e. *gen*) often smaller in length (in Figure 9 (a)). Sometimes, developers directly copy error message and search with that in search engine (see debug query example in Table 9). This type of query are too specific and often intended for few web documents (e.g., SO post discussion if exist). Thus, for *debug* query developers often search for smaller duration (smaller median in Fig. 9 (b)) and visit smaller number of websites (smaller median Fig. 9 (c)). In contrast, general code query (i.e. *gen*) often required lesser time - Fig. 9 (b) and web visit - Fig. 9 (c). In other word, among code queries, GPSEs are better at locating general code issues compared to other types (i.e. debug, testing , etc.). We

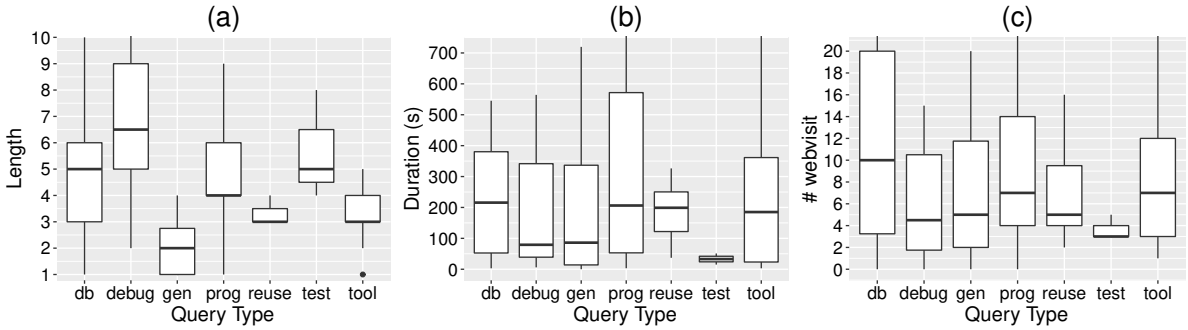

Figure 9: Properties for different types of code queries

plan to do a detailed analysis for such different type of code queries in future.

6 STUDY IMPLICATION

6.1 Implication of Code Intent Analysis

Here we discuss some areas where the code intent analysis can be leveraged.

Search System: It is important for the search engine to understand the search intent of users. Depending on the search intent returned results and other interaction with the search engine might vary. Search engines often use many meta information such as cookies, previous search history, URL click to understand users' search intent. This metasearch information is expensive to collect and not always available. For instance, the user might disable browser's cookie and history tracking or issue their first query. However, the model can predict a query intent on the fly and only requires the **query text**. So, this code intent model can be used as a complementary tool which can be plugged into any existing search system. Further, search engine often suggests related queries to the issued (initial) query. Code intent model can be used to guide this query recommendation process. If the user shows initial code intent for a search, higher code intent queries can be suggested.

Generalization: To score a query or sentence, our model requires only a set of the domain-specific token (S). In this paper, to achieve code intent we leverage general code related token from Stack Overflow [34]. This token set (S) can be easily extended or modified to identify specific code task. For example, "debugging" related tokens can be leveraged to predict whether the query intended for any code debugging task. Thus our model can be used to facilitate any further research where a fine granular code classification (i.e. debug, testing, etc.) is required. Additional knowledge about tags can be incorporated into the code intent analysis. For example, programming languages (e.g., java, haskell) or related technologies (e.g., react.js, mysql) can be assigned with constant scores. This process helps to mitigate any popularity bias of tags of the similar kind (e.g., java, haskell).

In addition to predicting query intent, the model can be applied to score any document (i.e. sequence of tokens). Developers can leverage *codeness score* to guide their document writing (e.g., API documentation) to make it discoverable by the search engine for code related search.

6.2 Implication of Empirical Findings

Unlike general non-code search, code issues usually require much more consultation with different documents including text (e.g., API documentation) and code (e.g., source code, bug reports), as evident by our findings that developers need to query more to complete a task (RQ3). Thus, code search imposes unique challenges for search engines when they treat documents with mixture of code, and text similarly as general textual document (e.g., news article). Thus, it is important for a search engine to incorporate effective retrieval models for code-mixed document and apply them effectively during code search.

We also report in RQ1 that code queries are more verbose and contain less vocabulary than non-code queries. Our code-intent model can be further trained with such characteristics, which can eventually impact GPSE's search performance. For example, if code intention is known by GPSE, it can restrict its search space. GPSE can also leverage the frequent added and deleted terms in advance from code queries to anticipate users' intent and recommend related queries accordingly.

7 RELATED WORK

There is substantial evidence in the literature to support the premise that developers use general purpose search engines during software development (e.g., [16, 22, 31, 36, 39]). Sim et al. [31] conducted a comparison study on various code search techniques of developers and observed that the general-purpose engine work better to find *reference examples* than do dedicated code search tools. Though code-specific search engines worked better in searches for subsystems, the general-purpose search engine, Google, worked better on searches for *blocks of code*. Furthermore, code-specific search engines (i.e. Koder and Krugle) perform better when searching for *subsystems of code* [31]. In a survey, Stolee et al. [36] found that 85% of developers search the web for source code at least weekly. This result is echoed in a survey by Hucka et al. [16], which found that 93% of the developers search on general-purpose web search engines for "ready-to-use" software and 91% of them search for source code. Additionally, Hucka et al. found that only 18% of participant developers use specialized code search engines for source code search.

Search logs of general-purpose search engines have been analyzed to identify different general query characteristics and users

search behavior (e.g., [30]). Similarly, dedicated code search engines' logs have also been investigated to determine the use of code search engines, topics of search queries, and format of queries (e.g., [2, 5, 28]).

In contrast, we analyze a search log of a general-purpose search engine (i.e. Google) which consists of both code and non-code related searches. We analyze query characteristics and users' search behavior for both code and non-query and explore the difference between them. Additionally, we evaluate the performance of the search engine for code related search compared to non-code general search.

In a study on Google developers, Sadowski et al. [28] observed that developers frequently search for code, conducting an average of five search sessions with a total of 12 queries in each workday. They also determined that programmers search to support a variety of information needs, such as looking for API examples, code understanding, debugging, or locating code snippets. The extensive use of code search engines in software development indicates that code search tools have a significant impact on developers' performance in practice [28].

In a large-scale survey, Xia et al. [39] identified the frequency and difficulty of different software development related web search tasks. They classify frequent search tasks including exceptions/error handling, reusable code snippets, programming bugs, and third-party libraries. On the other hand, search tasks including performance, multi-threading, and security bugs, database optimization, and reusable code snippets are identified as most difficult search tasks. They also observed that developers are likely to use general-purpose search engines (i.e. Google) to search for code.

Martie et al. [22] found that iterative support in search engine can provide better experience on searching for some specific development tasks. They categorized developers' responses to a question about why they search and observed that developers often search for code to implement a feature, support a design decision, or meet problem specification. This work is complementary to previous studies in its focus on comparing and contrasting code-related and non-code-related search tasks.

8 THREATS TO VALIDITY

Internal Validity. There might be some tokens (e.g., newly published library and API name) that carry code intent but are not included in our code token set. This can lead to incorrect *codeness score*. But developers often use some other tokens to describe such unknown terms when they search. For example, in query "telerik raddataboundlistbox winrt", although the second token is not included in our tag list, our classifier still identifies it as code related query by leveraging other tokens ("telerik" and "winrt").

Our tags popularity measure would assign a higher score for the query "java iterate array" than "haskell iterate array", as the programming language tag "haskell" is less popular than "java" on SO. Nevertheless, one could argue that the *codeness score* of both queries should be the same. However, in our case, separating code from non-code queries, such scenario is unlikely to occur between a code and a non-code query .

We use an automatic classifier to separate code and non-code queries. However, the classifier might make mistake and that may impact our analysis. To mitigate this threat, we select a *codeness score* threshold where the classifier achieves a better trade-off of *precision*, *recall*, and *F1-score* on manually annotated queries. However, due to the ambiguous nature of the query, it is nearly impractical to build a classifier which is absolutely accurate. Even we observe that the inter-annotator agreement is 0.85 (i.e. Cohen's Kappa Coefficient). This indicates that our automatic classifier effectively resembles human annotators.

External Validity. User's prior knowledge about a topic may impact the search performance of search engines. For example, a senior programmer might have more knowledge about a certain development task and use a search engine to refresh their knowledge. In contrast, the scenario for a new developer might be different. In our search log dataset, we do not have access to users' details information except anonymous ID. So, we treat all users similarly.

Further, we manually annotate 178 code queries into further categories (i.e. debug, general, etc.). This number of queries might not be adequate to come to a definite conclusion about their search characteristics.

We only study Google search log. Other GPSE may perform differently. However, since Google is the most popular GPSE, we think our study can be representative of different code and non-code query behavior.

Construct Validity. We use a search log which was collected using a Google Chrome activity tracker plugin. One inherent limitation of such tracker is that it tracks all the browser activity regardless of whether it's a search activity or not. For example, there might be some cases where a user searches for something and promptly switch tab to visit other websites (e.g., email, social media, etc.) and again comes back to the search activity. In such cases, some website might be incorrectly extracted as *clicked URL* for a query. However, such occurrences are usually less in number and can happen for both code and non-code related searches. So our comparison study (e.g., code vs non-code duration, number of web visit) would be less impacted by these threats.

9 CONCLUSION

Developers often use general-purpose search engines which are usually not optimized for code related search. We explore whether such choice is optimal for code related search by analyzing a search log consisting of both code and non-code query. Firstly, we build an automatic classifier that identifies code and non-code query.

We find that query characteristics (i.e. length) vary for code and non-code. We also observe that code related searching often requires more effort (i.e. time, web visit, query modification, etc.) than general non-code search. We further discuss how our study can be leveraged to improve code search using general-purpose search engines.

ACKNOWLEDGEMENTS

This work is sponsored by the National Science Foundation (NSF) grant CCF-16-19123 and CNS-16-18771. The conclusions of the paper are of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of NSF.

REFERENCES

[1] 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 *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*. ACM, 681–682.

[2] Sushil Krishna Bajracharya and Cristina Videira Lopes. 2012. Analyzing and mining a code search engine usage log. *Empirical Software Engineering* 17, 4-5 (2012), 424–466.

[3] Veronika Bauer, Jonas Eckhardt, Benedikt Hauptmann, and Manuel Klimek. 2014. An exploratory study on reuse at google. In *Proceedings of the 1st international workshop on software engineering research and industrial practices*. ACM, 14–23.

[4] Bing. [n. d.]. Bing Search. https://www.bing.com. ([n. d.]).

[5] 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)*. ACM, New York, NY, USA, 1589–1598. https://doi.org/10.1145/1518701.1518944

[6] Codealike. [n. d.]. Codealike. https://codealike.com. ([n. d.]).

[7] Cohen's Kappa Coefficient. [n. d.]. Cohen's Kappa Coefficient - Wikipedia. https://en.wikipedia.org/wiki/Cohen%27s_kappa. ([n. d.]).

[8] Christopher S Corley, Federico Lois, and Sebastian Quezada. 2015. Web usage patterns of developers. In *Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on*. IEEE, 381–390.

[9] Russ Cox. [n. d.]. Regular Expression Matching with a Trigram Index. https://swtch.com/~rsc/regexp/regexp4.html. ([n. d.]).

[10] 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 *Proceedings of the 2008 ACM symposium on Applied computing*. ACM, 1151–1157.

[11] Google. [n. d.]. Google Code Search - Deprecation Announcement. http://googleblog.blogspot.com/2011/10/fall-sweep.html. ([n. d.]).

[12] Google. [n. d.]. Google Search. https://www.google.com. ([n. d.]).

[13] 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 *Software Engineering (ICSE), 2013 35th International Conference on*. IEEE, 842–851.

[14] Vincent J Hellendoorn and Premkumar Devanbu. 2017. Are deep neural networks the best choice for modeling source code?. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. ACM, 763–773.

[15] Reid Holmes and Gail C Murphy. 2005. Using structural context to recommend source code examples. In *Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on*. IEEE, 117–125.

[16] Michael Hucka and Matthew J Graham. 2016. Software search is not a science, even among scientists. *arXiv preprint arXiv:1605.02265* (2016).

[17] Krugle. [n. d.]. Krugle Search. http://opensearch.krugle.org. ([n. d.]).

[18] Otávio Augusto Lazzarini Lemos, Sushil Krishna Bajracharya, and Joel Ossher. 2007. CodeGenie:: a tool for test-driven source code search. In *Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion*. ACM, 917–918.

[19] Otávio AL Lemos, Adriano C de Paula, Felipe C Zanichelli, and Cristina V Lopes. 2014. Thesaurus-based automatic query expansion for interface-driven code search. In *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 212–221.

[20] Erik Linstead, Sushil Bajracharya, Trung Ngo, Paul Rigor, Cristina Lopes, and Pierre Baldi. 2009. Sourcerer: mining and searching internet-scale software repositories. *Data Mining and Knowledge Discovery* 18, 2 (2009), 300–336.

[21] English Stopword List. [n. d.]. "http://www.lextek.com/manuals/onix/stopwords1.html". ([n. d.]).

[22] Lee Martie, André van der Hoek, and Thomas Kwak. 2017. Understanding the Impact of Support for Iteration on Code Search. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2017)*. ACM, New York, NY, USA, 774–785. https://doi.org/10.1145/3106237.3106293

[23] Collin McMillan, Mark Grechanik, Denys Poshyvanyk, Qing Xie, and Chen Fu. 2011. Portfolio: finding relevant functions and their usage. In *Proceedings of the 33rd International Conference on Software Engineering*. ACM, 111–120.

[24] Haoran Niu, Iman Keivanloo, and Ying Zou. 2017. Learning to rank code examples for code search engines. *Empirical Software Engineering* 22, 1 (2017), 259–291.

[25] Most Popular Programming Languages of 2017. [n. d.]. Top 100 programming languages. https://fossbytes.com/100-most-popular-programming-languages/. ([n. d.]).

[26] Mukund Raghothaman, Yi Wei, and Youssef Hamadi. 2016. SWIM: synthesizing what I mean: code search and idiomatic snippet synthesis. In *Proceedings of the 38th International Conference on Software Engineering*. ACM, 357–367.

[27] Steven P Reiss. 2009. Semantics-based code search. In *Proceedings of the 31st International Conference on Software Engineering*. IEEE Computer Society, 243–253.

[28] 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*. ACM, 191–201.

[29] SearchCode. [n. d.]. SearchCode Search. https://searchcode.com. ([n. d.]).

[30] Craig Silverstein, Hannes Marais, Monika Henzinger, and Michael Moricz. 1999. Analysis of a very large web search engine query log. In *ACm SIGIR Forum*, Vol. 33. ACM, 6–12.

[31] Susan Elliott Sim, Medha Umarji, Sukanya Ratanotayanon, and Cristina V Lopes. 2011. How well do search engines support code retrieval on the web? *ACM Transactions on Software Engineering and Methodology (TOSEM)* 21, 1 (2011), 4.

[32] Renuka Sindhgatta. 2006. Using an information retrieval system to retrieve source code samples. In *Proceedings of the 28th international conference on Software engineering*. ACM, 905–908.

[33] StackOverflow. [n. d.]. StackOverflow. https://code.openhub.net/. ([n. d.]).

[34] StackOverflow. [n. d.]. StackOverflow. https://stackoverflow.com/. ([n. d.]).

[35] StackOverflow. [n. d.]. StackOverflow Post - 46153155 . https://stackoverflow.com/questions/46153155/apply-function-to-all-pairs-efficiently. ([n. d.]).

[36] 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.

[37] Suresh Thummalapenta and Tao Xie. 2007. Parseweb: a programmer assistant for reusing open source code on the web. In *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering*. ACM, 204–213.

[38] Medha Umarji, Susan Sim, and Crista Lopes. 2008. Archetypal internet-scale source code searching. *Open source development, communities and quality* (2008), 257–263.

[39] 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 (01 Dec 2017), 3149–3185. https://doi.org/10.1007/s10664-017-9514-4

[40] Yahoo. [n. d.]. Yahoo Search. https://www.yahoo.com. ([n. d.]).

[41] Yunwen Ye and Gerhard Fischer. 2002. Supporting reuse by delivering task-relevant and personalized information. In *Proceedings of the 24th international conference on Software engineering*. ACM, 513–523.