How Developers Search for Code: A Case Study

Caitlin Sadowski
Google
Mountain View, CA, USA
supertri@google.com

Kathryn T. Stolee
Iowa State University
Ames, IA, USA
kstolee@iastate.edu

Sebastian Elbaum
University of Nebraska
Lincoln, NE, USA
elbaum@cse.unl.edu

ABSTRACT

With the advent of large code repositories and sophisticated search capabilities, code search is increasingly becoming a key software development activity. In this work we shed some light into how developers search for code through a case study performed at Google, using a combination of survey and log-analysis methodologies. Our study provides insights into what developers are doing and trying to learn when performing a search, search scope, query properties, and what a search session under different contexts usually entails. Our results indicate that programmers search for code very frequently, conducting an average of five search sessions with 12 total queries each workday. The search queries are often targeted at a particular code location and programmers are typically looking for code with which they are somewhat familiar. Further, programmers are generally seeking answers to questions about how to use an API, what code does, why something is failing, or where code is located.

Categories and Subject Descriptors

D.2.2 [**Software Engineering**]: Design Tools and Techniques—*user interfaces*

Keywords

Developer tools, code search, user evaluation

1. INTRODUCTION

Code search has been a part of software development for decades. It is unclear when it became entrenched in software development practices, but one of the first studies on the subject by Singer et al. reported in 1997 that the most frequent developer activity was code search [29]. The search mechanisms have evolved since that report, from "grep"-like tools to specialized and more efficient code search engines (e.g., [14, 21, 22, 25]). Perhaps more impressive is the increased scope of code search as repositories containing tens of billions of lines of code become available (e.g., [5,6,8,30]). Throughout this evolution, code search appears to have cemented its role in software development [3,17,28,31].

In spite of the pervasiveness of code search, we do not quite understand many aspects of how it is performed in-vivo. There is evidence, discussed in more detail in Section 2, that code search is practiced often, that there is a great deal of variation in how much effort is involved in code search, and that the developer's experience matters. Surprisingly, we still do not know much about the context in which code search is performed, what kind of questions developers are trying to answer when searching for code, what the most common search patterns are, and what patterns are predominant in certain contexts. Understanding how developers interact with code search is crucial to drive the next generation of techniques and tools to support searching, and future software development environments and workflows.

Through this work we provide a characterization of how developers at Google search for code, obtained through a combination of survey (396 responses) and search log-analysis (tens of thousands of records) generated by 27 developers during a period of 2 weeks. By carefully intertwining both methodologies, deploying the surveys just before a search begins, and having access to anonymized but precise user identification, we have been able to derive interesting and unique insights about code search practices. More specifically, the study finds:

- Programmers use code search to answer questions about a wide range of topics, including *what* code does, *where* is code instantiated, *why* code is behaving a certain way, *who* was responsible for an edit and *when* it happened, and *how* to perform a task. (RQ1)
- Code search tools are becoming so entrenched in software development that they are used to navigate even through code developers know well. (RQ2)
- Almost a third of searches are incrementally performed through query reformulation. Most queries were scoped to a subset of the code repository, with short sessions (lasting a median of 90 seconds) consisting of one or two queries. (RQ3)
- On average, developers compose 12 search queries per weekday (median is 6). Compared to previous literature [29], it would seem that code search is far more common than it once was. (RQ4)
- Search patterns vary greatly among contexts. For example, sessions with more clicks are typically associated with code reviews, and programmers searching

ESEC/FSE'15, August 30 – September 4, 2015, Bergamo, Italy
ACM. 978-1-4503-3675-8/15/08
http://dx.doi.org/10.1145/2786805.2786855

for code attributes are less likely to click on a result. (RQ5)

These findings have several important implications for the future of code search techniques and tools. First, given the pervasiveness of code search in software development, the effectiveness and efficiency of code search tools is likely to have an increasingly significant impact on developers' overall practices and performance. Second, code search is used by developers to answer diverse questions that require enriching its query format, its ranking algorithms, and its sources of data to better match those questions. Third, although code repository growth seems to be boundless, code search tends to have a well targeted scope. Search tools could benefit from inferring such scope by using, for example, a developer's code history to prioritize the results. Fourth, code information needs lead to different usage patterns that may be better served if code search tools were aware of those needs by, for example, providing additional operators or being cognizant of other tools currently being used by the developer. Last, as search sessions become shorter and more focused, there is a greater incentive for code search tools to be better integrated into software development environments to reduce the number context switches.

2. RELATED WORK

Code search can take on many forms, from searching for code reuse to searching for concept localization. Efforts to observe code search have ranged from surveys to lab studies, log analysis to observations of work practices. In this section, we focus on studies that explore code search behavior and existing code search tools.

2.1 Previous Code Search Studies

Code search has been studied in the wild by observing programmers directly [29] or analyzing search logs [3], in controlled settings using pre-defined tasks [2, 3, 12, 15, 17, 28] or by surveying developers about their search practices [27, 31]. In some cases, the goal was not to observe code search directly, but rather code search was observed as part of information seeking activities performed by developers [3, 12, 15]. In these search studies, participants come from companies [2, 15], student populations [12, 17, 28, 31], or crowdsourcing platforms such as Amazon's Mechanical Turk [31] or online newsgroups [27].

Surveys have reported that 50% of programmers search for code frequently and 39% search occasionally [28]. Furthermore, among developers who program daily, 59% search for code daily [31]. In addition to frequency, surveys can also inform search motivations. In one survey, the criteria that guide the final code selection were reported [27]. Among the 69 respondents, 77% consider the functionality, 43% consider the licensing terms, and 30% consider user support when selecting code for reuse. Among search sessions dealing with java, 34.2% had a goal of finding/learning about an API [10]. Of those, 17.9% included the word 'sample' or 'example' in the query. Surveys often suffer from response bias where subjects' recollections of past events might not match actual past events. By placing the survey directly before a search occurs, the response bias in our study is mitigated as the programmer does not need to reflect on past activity, but rather report on current activity. In our methodology, we further combine survey analysis with log analysis to bring context to the search behaviors. We revisit prior survey results in Section 4.1.

A prevalent finding across observational studies is that code search is common among maintenance tasks. Observing eight programmers in a company across several sessions revealed that 100% of programmers perform search activities using standard tools such as grep or in-house toolsets [29]. With a more focused study of one programmer, that same study revealed that a search task is performed on 57% of the days in which a programmer works, and that search is the most frequently-performed activity [29].

Lab studies are often designed to observe behaviors other than search, but search activities are recorded. While performing program maintenance tasks, 92% of participants, all corporate employees, used search tools [15]. When starting maintenance tasks, student programmers typically start with search within an IDE (40 / 48 participants), and occasionally by searching the Web using Google [12]. Other studies have shown that 100% of programmers use search during program maintenance tasks [17].

Search terms are also the focus of some lab studies. One study [26] found that average searches to Google contain 4.7 terms versus 4.2 terms for the now deprecated external Google Code Search, 4.1 for Krugle [14], 3.7 to Koders [13], and 3.8 to SourceForge [30]. For the first query of a session, 48% of queries contain just code, 38% contain just natural language, and 14% contain a mix [3]. We revisit these observations in the context of our search logs in Section 4.3.

Yet other studies relate search behaviors with experience. Brant et al. found that when a programmer wants to learn a new concept, each web session took tens of minutes [3]. In contrast, when a programmer wants to refresh their memory of a concept, like a syntactic detail, a web session took tens of seconds. Clarifying implementation details usually took approximately one minute. We revisit these observations in the context of our search logs in Section 4.5.

Beyond just the use of search, some lab studies have characterized search sessions. For example, in two 2-hour laboratory experiments, Li, et al. found that developers conducted 4.3 or 6.6 web searches, on average [17]. Another controlled study found that each search session involved 2.38 queries [26]. For tasks taking an average of 125 seconds to complete, Hoffman, et al. observed an average of 1.9 queries to Google [10]. These studies indicate a range of frequencies for search activity, but the observation remains that search is a frequent activity among programmers. Rather than just using lab studies, using search logs can allow researchers to analyze more data, providing useful insights into the frequency of use for tools as well as which behaviors and features are most common. For example, when analyzing search logs and identifying sessions based on IP address and durations of inactivity, it was found that users perform an average of 1.45 queries per session [3]. However, when using logs some of the details of the search process, like the user's context, are masked. For these reason, we combine log analysis with in situ surveys. We revisit these findings in Section 4.4.

2.2 Existing Code Search Tools

Several code search engines exist today. Sim's paper [28] compares the following: Koders [13] aka ohloh [22], Google (general), Google Code Search (the former public version), Krugle [14], SourceForge [30], exploring the number of Java

files indexed, retrieval algorithm, whether regular expressions are permitted in the query, and the presence of special features such as filters. Each of these search engines is publicly available and indexes based on keyword and/or method signatures. Other source code search engines have started to provide more semantic features. Exemplar uses a keyword search that ranks based on application descriptions, APIs used, and the data flow among those APIs, providing a more semantic search approach [9, 20]. S6 uses keywords, type signatures, and a test-based approach to find modules that match the keywords, have a specified interface, and behave as specified using an example [23]. CodeGenie uses a test-driven approach to search for a reuse code [16]. Sourcerer has indexed over 38 million lines of Java code, maintaining a database with keyword, structure, data type, and pattern information [1,18]. Satsy uses a unique query model consisting of input/output pairs and an SMT solver to find source code that behaves as specified by the query [31].

3. STUDY

Most software development at Google is done in one giant repository, and developers are able to browse that repository to review (and potentially use) code outside of their specific project [24]. Google engineers are able to search over the internal codebase using an internal code search engine; this tool is similar to the public code search tool that was available externally from 2006 until 2012 [7] and a variant of this code search tool is currently available externally for Chromium [4]. Developers can search for code with regexes and operators such as `lang:`. For example, `lang:java server` could return a result like `MyServer.java`. Queries with the `file:` operator match against the file path. Operators may also be used to return results without a certain property, so that a path supplied with `-file:` will return results that do not match the path. In addition to returning a ranked list of files as search results, the code search tool also surfaces some additional metadata about the files, and allows developers to navigate through source code files via the directory structure or by clicking on references (e.g. to jump to a definition or usages).

3.1 Data collection

We experimented with different study designs in order to capture the questions that developers were trying to answer when searching and fill in some gaps in existing literature. In the end, we developed a lightweight survey methodology based on a browser extension. We combined this with logs analysis. In total, we invited 40 developers, from which 27 agreed to participate from 23 different teams throughout Google over a period of two weeks. These developers were known by at least one of the authors and were deemed potentially responsive in the time frame when the study was conducted. The participants included 18 software engineers, 8 software test engineers, and one release engineer. There were 22 males and five females, nine with B.S. (or similar degree), 10 with an M.S., and eight with a Ph.D., with an average of 9.75 years of post-degree software development experience, and 3.4 years at Google.

3.1.1 Surveys

We wrote a browser extension that directed developers to our survey when they accessed the internal code search site, and had participants install this extension on all computers they used while working. Through this extension, we collected survey results before a search started. So as not to overwhelm developers or repeatedly ask about the same search session, we configured the extension to survey a maximum of 10 times a day (per browser run), and only when at least 10 minutes passed since last code search activity. We asked developers to fill in the survey every time it appeared, although we observed that sometimes developers would skip filling it out (particularly when working at their desk with a second developer). All told, we collected 394 survey responses.

The full text of the multi-select survey questions can be found in the first column of Table 2. We wanted to keep it brief; from our previous experience with deploying similar surveys we found that 3-4 questions maximizes response rates. Developers could select multiple options in the multiple choice questions.

We refined the survey questions and answers through 2 pilot iterations with 2-3 developers. These pilot iterations helped us refine the questions, and adjust the survey frequency to balance data capture with a load that developers deemed acceptable. The developers involved with the pilots were not included in the final data logs.

3.1.2 Logs

The logs collected search events and interactions with the code search tool. When a developer goes to the code search site, all requests to the code search servers are logged. The logs included the user, time, search terms, results clicked, files viewed, as well as a unique per-tab ID. Since one user action (e.g. searching for the term `foo`) results in many logged events, some entries are associated with loading different parts of the code search UI. The 27 developers observed over the two week period generated 180,429 entries. We filtered out all logs that were not related to a user event (e.g. clicking the search button or selecting a result). We further filtered the logs to remove duplicate entries, resulting in 77,632 events.

We analyzed these logs using search *sessions*. A search *session* represents when a developer goes to the code search site, performs one or more searches, and potentially also clicks on results or views files. Sessions were computed as activities in one or more browser tabs by a single developer with a 6-minute timeout between events. This 6-minute timeout was derived empirically, in part following prior work [3].

After filtering, we ended up with 1,929 sessions. Of these, 1,429 sessions contained a search event. Of those with no search, 476 (95%) were the result of browsing the directory information in the search interface, without an actual search query. The remaining sessions either had zero duration (i.e., open the search interface and then close it) or had other patterns of behavior that could not be discerned into recognizable patterns.

3.2 Research Questions

We investigated 5 research questions.

- RQ1: Why do programmers search?
  We performed a qualitative analysis on the freeform responses to the fourth survey question (see Table 2), "What question are you trying to answer?", to identify a set of question topics. This was done by open coding the responses to identify themes.

- RQ2: In what contexts is search used?
  We explore context from three angles: what activities developers are performing when searching, what they are trying to learn, and how familiar they are with the code on which they are searching. We utilize the data collected through the first three survey questions (see Table 2),.
- RQ3: What are the properties of the search queries?
  We explore the keyword queries in terms of the number of words, the use of reserved operator words, and patterns of query reformulation. We utilize the search logs to answer this question.
- RQ4: What does a typical search session entail?
  We answer this question by quantifying how often and for how long developers search for code, and how many search results they explore. We utilize the search logs to answer this question.
- RQ5: Do different contexts lead to distinct search patterns?
  We analyze the survey and logs jointly, identifying search patterns that differ across contexts.

4. RESULTS

In this section, we present our analyses and results for each research question.

4.1 RQ1: Why do programmers search?

To answer this question, we analyzed the questions programmers were trying to answer by coding their textual responses to the fourth survey question (Table 2).

We wanted to understand why developers search and what questions developers were trying to answer with code search. To address this question, we had a fill-in question on the survey "What question are you trying to answer?". In all, we had 266 responses to this question. We eliminated 7 responses where the developers stated they were using the code search interface to navigate to a different task and not trying to answer a question. This resulted in 259 different responses. We wrote each response down on a card, and then performed an open card sorting [11] exercise with three sorters. We kept going until we reached a fix point of categories. We then identified higher-level concepts from the card sorting to organize the developers' activities. The full breakdown is in Table 1; this table contains the count, percentage, and name for each category, as well as a representative sample of the searches in each category. The high-level categories are bolded. These roughly correspond to what type of question was being asked, whether it was about *how* to do something, learning *what* code does, determining *why* code is behaving as it is, locating *where* something is, or finding out *who* did something and *when* they did it.

Example Code Needed (How).

The most common theme deals with getting specific API or library information or functional example, representing over a third (33.5%) of the surveys. This is strikingly similar to prior work that found 34.2% of search sessions had a goal of finding/learning about an API [10]

Responses in the **discover correct library for task** were focused on trying to find a library that would fulfill a particular specific task, e.g. that transform known inputs into desired outputs. In contrast, **API consumer needs help** encompasses situations where the developer knows the name of the specific class or API they want to use, but are trying to figure out what methods or parameters to use. These kinds of questions could also potentially be answered by viewing documentation for APIs. Searches in the **example to build off of** category are looking for a specific function. **How to do something** covers examples where a developer is trying to figure out how to complete a broad task, as opposed to finding or using a specific API.

Exploring or Reading Code (What).

Representing 26% of the survey responses, this category deals with exploring code and reading code. In contrast to the where, these activities are more exploratory and less targeted. **Browsing** encompasses situations where a developer is not trying to answer a specific question, but is instead navigating to and reading some code. **Check implementation details** deals with situations where a developer is trying to understand how a particular function is implemented, or sometimes navigating to very specific place in the code base to check a specific detail such as the exact name of a flag (16 out of the 51 instances are focused on answering these types of extremely targeted queries). Responses in the **name completion** category were searches where the developer could only remember part of the name of a class or file they wanted to reference, and were searching to find the entire name. **Check common style** represents situations where the developer is checking how something is usually done in the repository.

Code Localization (Where).

This category focuses on finding a particular location in source control, representing 16% of the searches. **Reachability** encompasses situations where a developer is trying to trace through source code to find out how pieces of code are connected; for example, where specific types are used or where the call sites (or definitions) of specific methods are. **Showing to someone else** deals with searches that are focused on showing details known to the searcher to another colleague. Responses in the **location in source control** category were just trying to check the specific location of a known object in the depot. For example, checking the location of a file when the name is known.

Determine Impact (Why).

Understanding the impact of a change or why code behaves a certain way represents 16% of the searches. **Why is something failing** searches encompass situations where the developer is trying to understand why there is a mismatch with how they think code should behave, and how code is actually behaving. **Side effects of a proposed change** represents searches focused on verifying whether assumptions made in a changelist are valid. In contrast to reachability, **understanding dependencies** deals with specific queries about how projects are connected, e.g. by the build system.

Metadata (Who and When).

Answering questions related to how other developers have interacted with the depot represents 8% of the categories. **Trace code history** encompasses situations where developers want to find out when something was introduced or

Table 1: Categories of answers to fourth survey question, "What question are you trying to answer?"

Category	Example	Count	Percent
Example Code Needed (How)		**87**	**33.5%**
API consumer needs help	"I want to know how a function should be called."	57	22%
Discover correct library for task	"Best way to convert output stream into a string of limited length."	14	5%
Example to build off of	"Just want to copy-and-paste some code I'm changing"	9	3.5%
How to do something	"How to write a hash function"	7	3%
Exploring or Reading Code (What)		**67**	**26%**
Check implementation details	"What a particular script does"	51	20%
Browsing	"Re-familiarizing myself with some code referenced in a CL under review."	11	4%
Check common style	"Where are friend classes usually declared?"	3	1%
Name completion	"I'm looking for an enum member that I think begins with a particular prefix."	2	1%
Code Localization (Where)		**41**	**16%**
Reachability	"Where a class is instantiated"	22	8.5%
Showing to someone else	"I'm trying to create a link to a known piece of code, to provide to someone else."	10	4%
Location in source control	"Where are all the boxed environment configurations?"	9	3.5%
Determine Impact (Why)		**42**	**16%**
Why is something failing	"I'm wondering why a CL I wrote earlier did not fix a currently-occurring production problem and am reading the code to diagnose."	26	10%
Understanding dependencies	"Looking for dependencies of a build file"	12	4.5%
Side effects of a proposed change	"Am I about to blow up production with my CL"	4	1.5%
Metadata (Who and When)		**22**	**8.5%**
Trace code history	"Who last touched some code."	13	5%
Responsibility	"Who is allowed to approve changelists for a particular file."	9	3.5%
Total		259	100.00%

changed. **Responsibility** represents situations where a developer is trying to establish ownership of a particular piece of code. Ownership is important when deciding who should code review a change to a particular project, or who to talk with about a proposed change.

Overall, we find that code search is about more than just finding examples for how to do something, though getting API information and examples is the most frequent task. Many of the surveys indicated that the programmer was just reading code or exploring the history or code attributes to understand who did or can do something and when. Over 16% of the surveys were concerned with finding a particular location in the code, also referred to concept location [19].

4.2 RQ2: In what contexts is search used?

To address this question, analyzed the multiple-choice survey questions using summary statistics.

4.2.1 Purpose for Searching

The collected data from the survey's first three questions is summarized in Table 2. The table includes a count and percentage for each potential answer within each question.

Overall, we see that code search is utilized across many development activities performed by a typical software engineer at Google. Most searches are performed while working on a code change (39%). The activities of triaging a problem or reviewing a change review also use code search. Note that the distribution does not necessarily mean that code search is more prevalent in one particular activity, but could also reflect the amount of time spent in each of these activities by a developer.

With respect to the scope of the search, most searches focus on code that is familiar or somewhat familiar to the developers (17% and 44% respectively). This may indicate that code search tools are becoming so entrenched in software development that they are being adopted to navigate even through code that developers know well.

Almost half of the developers use code search to understand how code works (46%), and to a less extent how to use it (21%). One unexpected finding was the diversity of information that developers try to obtain through code search. We received 14% of responses that included other learning objectives, such as understanding build relationships, making a recommendation to a peer, or assessing tests that may be triggered.

4.2.2 Correlating Familiarity, Learning, and Doing

Table 3 provides a breakdown of the first level interactions among the survey responses across questions. This reveals some interesting patterns that were not obvious when analyzing the responses to individual questions. For each pair of response options the table provides a count and two percentages (%h conveys the percentage over the count of the response in the row, while %v conveys the percentage over the count of the response in the column – these percentages

Table 3: Survey results breakdown. We consider first level interactions among single predefined responses ("other" or multiple marked responses are not accounted). The cells contain the number of responses for the row and column answers, or the % using as denominator the occurrences of the row (%h) or the column (%v).

Doing and Familiarity	Very Familiar	%h	%v	Not Familiar	%h	%v	Somewhat Familiar	%h	%v	-	-	-
Exploring	5	11	7	20	43	14	12	26	7	-	-	-
Working on a CL	26	16	38	51	32	35	75	45	41	-	-	-
Triaging a problem	10	11	15	33	37	23	41	46	24	-	-	-
Reviewing a CL	10	16	15	22	35	15	29	47	17	-	-	-
Designing a new feature	4	17	6	6	25	4	5	21	3	-	-	-
Doing and Learning	Code attributes	%h	%v	How code changed	%h	%v	How code works	%h	%v	How to use code	%h	%v
Exploring	6	13	14	2	4	7	20	43	11	4	9	5
Working on a CL	16	10	36	7	4	23	54	34	31	42	26	53
Triaging a problem	9	10	20	12	13	40	45	50	26	10	11	13
Reviewing a CL	4	6	9	5	8	17	32	52	18	7	11	9
Designing a new feature	2	8	5	1	4	3	3	13	2	6	25	8
Familiarity and Learning	Code attributes	%h	%v	How code changed	%h	%v	How code works	%h	%v	How to use code	%h	%v
Very Familiar	4	6	9	9	13	30	26	38	15	3	4	4
Not Familiar	13	9	30	8	6	27	56	39	32	42	29	53
Somewhat Familiar	24	14	55	12	7	40	81	47	46	27	16	34

Table 2: Survey results. The survey included 394 responses from 27 software developers during a period of 15 days.

Question	Count	%
1. What are you doing?		
- Exploring	47	12%
- Working on a CL	159	**39%**
- Triaging a problem	90	22%
- Reviewing a CL	62	15%
- Designing a new feature	24	6%
- Other (e.g., finding a link, helping someone)	25	6%
2. How familiar are you with the code you are searching for?		
- Very Familiar (e.g., I wrote it)	68	17%
- Somewhat Familiar (e.g., I've seen it before)	173	**44%**
- Not Familiar	145	37%
- Not sure	6	2%
3. What are you trying to learn?		
- Code attributes (e.g. location, owner, size)	44	12%
- How code changed	30	8%
- How code works	175	**46%**
- How to use code	80	21%
- Other (e.g., build, configuration, test)	53	14%
4. What question are you trying to answer?		
⟨ free-text response ⟩	271	**100%**

may not add up to 100% as only single responses were considered and these were multi-select questions). For example, the cell in the second row and column indicates that 5 survey responses included "Exploring" to the first question and "Very familiar" to the second question. This corresponds to 11% of all the exploring activities reported, and 7% of all very familiar responses.

Although the same trends as before can be observed in Table 3, there are some new pieces of information worth highlighting. When looking at the responses to "Doing and Familiarity" we see that almost half of the work on CLs, triaging, or reviewing occurs on somewhat familiar code. Still, 35% of the time developers search for non familiar code when performing those tasks. Perhaps more interesting is that most searches for exploring and designing new features occurred on not familiar code. This seems to indicate that the scope of the search for new development versus maintenance activities is different.

For "Doing and Learning" we see that, independent of the activity, learning how code works is consistently desired by developers performing a search (∼50%) except for when designing a new feature when how to use the code is most common. On the other hand, developers performing a search when performing triaging most often asked how the code changed.

For "Familiarity and Learning", most search questions on very familiar code were about how code works. Developers seem to be using search support for navigating code they know but may not recall particular details. The scope of the search shifts in the context of how code changed which were performed on somewhat familiar code (40%), and it shifts even further to not familiar code when asking how to use code (53%). Just as in the case of what the developer is doing, there seems to be a clear relationship between the scope of the search and what the developer is trying to learn.

Taking the survey analysis a step future, we correlate the responses with search behavior patterns in Section 4.5.

4.3 RQ3: What are the properties of the search queries?

In this section, we use the search logs to explore the properties of a typical search query and series of queries with no result clicks between. We filtered out terms from the search that are typically placed there by default.

The average number of keywords per query was 1.85 (median of 1, maximum of 11). Compared to previous work, this number is rather low. Prior work looking at the number of keywords in searches to Google Code Search [7] reported 4.2 terms per query [26]. We conjecture that these changes may have resulted from search engine improvements, user familiarity with query formation, and also from the more incremental nature of search (discussed in the next section).

In addition to counting the terms per query, we can look inside the queries at the content. Of the 3,870 queries, 1,025

(26.5%) contained the operator `file:` followed by the name of a path expression in which the search should take place, for example, to restrict search to a particular project. This indicates a high frequency of targeted searches. A specific language was specified using the `lang:` operator for 210 (5.4%) of the queries.

Often, we observe a series of two or more searches with no result clicks in between. The final search may or may not have clicks. There are 970 pairs of search events in the logs with no clicks between (this is the `SS` pattern from Section 4.4), impacting 429 (30%) of the sessions.

Using Levenshtein's Distance[1] at the word level, we look at the distance between successive search queries. This metric counts the number of insertions, deletions, or substitutions required to change one sequence into another. As an example, a programmer may refine their query by adding an additional keyword, such as the name of a particular file where the desired code lives. Here, the first query would contain a word or sequence of words, and the second query would be equivalent to the first with the addition of a file name. That would cause a distance of 1. Modifying a word in a query also results in a distance of 1.

Of the 970 instances of `SS` within a session, the range of Levenshtein's distance was one to eight. Of those, 734 (76%) had a distance of one from the previous search query, indicating potential query reformulation. Also, 52 (7%) of the these involved adding restrictions to the file paths returned. The average time between search queries was 23 seconds (median was eight).

Overall, we see strong evidence of frequent query reformulation within the search sessions, and it typically happens within eight seconds of the original query. We can hypothesize that the typical user can scan results, determine if they meet their needs, and reformulate an initial query within eight seconds.

4.4 RQ4: What does a typical search session entail?

In this section, we use the search logs to explore the properties of a typical search session (activities by a single developer with a 6-minute timeout between events) in terms of duration, number of queries, number of result clicks, and the patterns of behavior.

Among all developers, the total number of sessions with a search event was 1,429. The typical search session lasted an average of 3 minutes 30 seconds (median was 1 minute 29 seconds), had 2.12 tabs (median 1.00), and conducted 2.64 search queries (median 1.00). Each session had an average of 3.41 result clicks (median 2.00). In our logs, the number of queries was higher than in prior work, which found an average of 1.45 queries per session [3]. The longest session lasted nearly 42 minutes, the most searches in a session was 67, the most result clicks in a session was 92, and the largest number of tabs was 50.

For all developers, there were an average of 53 total sessions (median 42). Developers performed an average of 140 searches over the course of the 15 days with a median of 86. Two of the 27 developers performed more than 500 searches, whereas the remaining performed fewer than 200 searches each. Of the searches, 96% were performed on weekdays compared to weekends. On the average weekday (4 of the 15 days were weekends), each participant wrote an average of 12 search queries (median is 6). While these numbers are from 2013, linking this to prior work, in 1997, Singer et al. found programmers search on 57% of the days in which they work [29]. Clearly, search is becoming an integral component of the modern software development process.

[1]This metric was chosen over Hamming distance because it relaxes the requirement on even lengths of the strings being compared.

Table 4: Patterns of Whole Sessions

Pattern	Meaning	Sessions	%
`SC`	1 click	264	18.5
`(S1)+`	1 result	189	13.2
`SCC+`	1 search 2+ click	171	11.9
`S+C+(S+C+)+`	2+@(search+ click+)	163	11.4
`S+`	0 clicks	127	8.9
`SS+C`	2+ searches, 1 click	67	4.7
`C`	Starts with a click	62	4.3
`SS+CC+`	2+ search, 2+ click	39	2.7
Other	—	347	24.3
Total		1429	

Within each session, we explored patterns of behavior related to the searches and the result clicks. To describe these sessions, we use `S` to denote a search event (queries are associated with these events), `1` to indicate the search returned only one result, and `C` to denote when the programmer clicked on a search result. The following sections describe the patterns observed across whole sessions as well as micro-patterns within sessions.

4.4.1 Whole Session Patterns

Representing each session as a series of searches and clicks, we observe several common patterns, shown in the *Pattern* column of Table 4, followed by a textual description in the *Meaning* column. The number and percentage of sessions are in the subsequent columns. For example, the pattern `SC`, which means there was one search and one click, represents 264 or 18.5% of the total sessions. The `+` is used like a regular expression, indicating one or more repetition. For example, `(S1)+` means that there was one or more searches that each returned exactly one result. A session with the pattern, `S1S1`, would match this category. The patterns were identified using manual analysis to find succinct, orthogonal representations of the whole sessions.

The *Other* category represents patterns that could not be concisely described. For example, the pattern, `S1SCCCC` is a compound pattern between `S1` and `SCC+`. The pattern, `SCCCCS`, starts with `SCC+` and ends with `S+`.

Overall, we observe that over 75% of the sessions involve one or more clicks. The no-clicks pattern may indicate that the programmer does not find what they want, or they find what they want by observing the previews in the results. One anomalous pattern that was common but did not start with search was `C`, involving just a single click. This pattern can emerge when a user opens a search result in a new tab at least 6 minutes after their last recorded search event.

4.4.2 Micro-Patterns in Sessions

While the eight patterns in Table 4 represent over 75% of the sessions, programmers often perform multiple searches in the same session, each with its own click patterns. For this reason, we capture micro-sessions as sequences of events delimited by searches. In total, there were 3,780 micro-sessions

Table 5: Patterns of Micro-Sessions

Pattern	Meaning	Micro-Sessions	%
S	1 search, 0 click	1466	38.8
SC	1 click	909	24.1
S1	1 result, implicit	599	15.7
SCC	1 search 2 click	219	5.8
SCCC	1 search 3 click	227	6.0
SCCC+	1 search 4+ click	202	5.4
Other	—	158	4.1
Total		3,780	

(one micro-session for each search). Table 5 shows the six most common micro-patterns, representing nearly 96% of the micro-sessions.

In Table 5, we see that nearly 39% of the searches are followed by zero clicks. There are two potential reasons for this. The first is that the search needs to be refined, as we see with any of the S+ patterns in Table 4. The other explanation is that the answer to the programmer's question can be found on the results page. Another common pattern is S1, where only one result is returned. This could be the result of query autocompletion, using a URL to navigate directly to results, or a very exact search term.

4.5 RQ5: Do different contexts lead to distinct search patterns?

Since we collected both logs and survey data of search activity using the same user base, we have an opportunity to identify deeper patterns of behavior based on the programmer's context. In this section, we link the survey information from RQ1 and RQ2 to the patterns identified in RQ4.

Every developer from whom we collected logs also submitted at least one survey. The average number of surveys per person was 14.5 (median 9), with a range from one to 69. We dropped 11 surveys that were filled out before we began collecting logs information.

We linked the surveys to search whole-sessions (RQ4) by associating the surveys with search sessions that were closest in time. The average time between a survey submission and a search was 40 seconds with a median of 11 seconds and a standard deviation of 2 minutes 47 seconds. This means that on average, the survey took 40 seconds to fill out and submit. In some cases, a survey was not associated with any search session (NS). This could happen, for example, if a developer is browsing directories or reviewing prior results.

Table 6 shows the relationships between the quantitative questions in the survey (Table 2) and the search session type (Table 4). In this table, S+C+ covers the session types of SC, SCC+, SS+C, and SS+CC+, (S1|(S+C+)){2,} is used to capture session type S+C+(S+C+)+ and Other sessions, and NS is used to identify surveys that are not grouped with a search session. This table shows the count and row percentage for each session type. For example, if a programmer indicated they were *Somewhat familiar* with code being searched for, the search pattern S+C+ followed 30% of the time and the pattern S+ followed 21% of the time.

A programmer who is *very familiar* with the code is most likely to follow the S+C+ pattern, representing 52% of the searches in that context. In eight (53%) of those sessions, there is exactly one search followed by exactly one click. This is rather intuitive, as they likely know where they want to go and use search to get there.

Regarding what the programmer wants to learn, if they want to learn about code attributes, 19% of the time no file is clicked, perhaps indicating that this answer is found in the previews on the search results page, plus another 25% of the time the search only returned one result.

One expectation we had was for developers who are *not familiar* with the code to follow a search with many searches and many clicks. We have some evidence of this, where 28% of the sessions match S+C+ and 24% match (S1|(S+C+)){2,}.

Table 7 links the categories from Table 1 to the whole search sessions in Table 4. As in Table 6, S+C+ covers the session types of SC, SCC+, SS+C, and SS+CC+, (S1|(S+C+)){2,} is used to capture session type S+C+(S+C+)+ and Other sessions, and NS is used to identify surveys that are not grouped with a search session. For example, of the surveys looking for *API consumer need help*, 38% were the session type S+C+.

Developers looking to understand the *Side effects of proposed change* had one or more searches followed by one or more result clicks 75% of the time. This same pattern, S+C+, represent 32% of the searches that want to *check implementation details*.

Developers with *Reachability* questions often result in a pattern of only viewing search results, without clicking through to a file (pattern S+ happens 30% of the time). Developers who want to *Discover library for task* only view search results 73% of the time (pattern S+). This represents eight sessions; seven (88%) have a single search and only one has multiple searches.

5. DISCUSSION

The combination of survey and log data has led to insights regarding where, why, and how programmers search for source code. In this section, we elaborate on the main findings, which include observations of developer behavior and implications for tool designers.

5.1 Observations of Developer Behavior

Developers search frequently. The average developer in our study had 5.3 search sessions on a typical weekday, performing 1-2 searches and clicking on 2-3 results per session. With this prevalence of search activity, greater than expected given previously reported results, search speed and recall/precision will impact a developer's productivity.

Developers search for examples more than anything else. Based on the free-text responses, the most common questions dealt with finding examples or sample code (34%). Despite the breadth of purposes code search serves, examples still need to be well supported.

Developers search local. Repositories are growing, but developers mostly search familiar or somewhat familiar code. Developers use search for quick navigation of a code base; over 26% of the queries restricted the file path to search within.

Queries are incremental. Compared to prior studies in a controlled lab environment that observed queries with an average of 3-5 terms [26], in our study the queries had just 1-2 terms, but were incrementally refined. These differences could be caused by the more sophisticated tools available (e.g., search auto completion), the experience of the developers involved who could better scope a search or know the standard naming conventions others may used, or by the richer development contexts with additional information.

Table 6: Relationship Between search whole-session patterns (Table 4) plus NS for "sessions" that contain no search event (e.g. browsing directories or reviewing prior results) and Familiarity, Activity, and Learning (Table 2). The cells contain the number of responses for the row and column, or the % using as denominator of the row (%).

What are you doing?	S+C+	%	(S1|(S+C+)){2,}	%	S+	%	(S+1)+	%	S+CS+	%	NS	%	^C	%
Answering a question	0	0	0	0	0	0	1	50	1	50	0	0	0	0
Designing a new feature	5	45	5	45	1	9	0	0	0	0	0	0	0	0
Exploring	6	26	7	30	3	13	5	22	1	4	1	4	0	0
Reviewing a CL	18	36	8	16	10	20	3	6	4	8	1	2	6	12
Triaging a problem	15	26	12	21	8	14	4	7	10	18	4	7	4	7
Working on a CL	32	33	18	19	15	16	12	13	8	8	7	7	4	4
Other	2	14	3	21	1	7	4	29	1	7	2	14	1	7
How familiar are you?	S+C+	%	(S1|(S+C+)){2,}	%	S+	%	(S+1)+	%	S+CS+	%	NS	%	^C	%
Not sure	0	0	0	0	1	50	1	50	0	0	0	0	0	0
Not Familiar	27	28	23	24	9	9	11	11	9	9	10	10	7	7
Somewhat Familiar	36	30	21	17	26	21	15	12	13	11	6	5	4	3
Very Familiar	15	52	3	10	2	7	2	7	3	10	1	3	3	10
What do you want to learn?	S+C+	%	(S1|(S+C+)){2,}	%	S+	%	(S+1)+	%	S+CS+	%	NS	%	^C	%
Code attributes	6	19	7	22	6	19	8	25	1	3	4	13	0	0
How code changed	2	17	2	17	1	8	1	8	2	17	1	8	3	25
How code works	34	31	22	20	9	8	17	15	14	13	8	7	7	6
How to use code	15	28	11	20	17	31	0	0	5	9	3	6	3	6
Other	21	50	8	19	5	12	3	7	3	7	2	5	0	0

Code familiarity does not mean less clicks. For *very familiar* sessions, the average number of result clicks was 2.5, slightly lower than the average for *not familiar* sessions, with an average of 3.1, but the difference is not significant ($p = 0.41$ using a Mann-Whitney non-parametric test of means). This is contrary to prior work that showed programmers who are less familiar with the code they are searching for are more likely to click more results [29].

5.2 Implications for Tool Designers

Focus on developer's questions. Search tools should start focusing on comprehension tasks based on the questions developers are asking and how they are asking them. They should also tap into the code repository metadata such as build dependencies, developers involved, file and workflow permissions, tests executed, and relevant review comments since many of the questions asked during search include such pieces of information.

Provide simple code examples. As shown in Table 1, 34% of the surveys indicated a desire to find code examples, for example, illustrating API usage. To reduce the duration of search sessions and keep developers productive, these examples should be minimal and illustrate common usage. Integrating the usage frequency for patterns of API usage into a ranking algorithm, or their size, could help developers find needed information faster.

Consider code location. Query features such as the `-file:` operator help support the specification of a search scope, but additional operators could help scope the search, for example, to code touched by specific developers or groups of developers (`-dev:`). Furthermore, tools should predict a developer's locality needs based on search history and the files recently touched to better rank the matches.

Consider richer context. Search patterns vary across activities. A tool cognizant of the contextual elements (e.g., applications open, recent communications) associated with different activities could be more effective. For example, if a developer is working with the reviewing and bug tracking tools, then the search tool could infer that triaging is happening and give priority to matches that show code changes.

Consider integrating code search into development environment. With search sessions lasting only a few minutes, and a small number of queries per session, the time to context switch between the development or code review environment and the search tool becomes more dominant. Integrating code search into the development environment could reduce that overhead.

6. THREATS TO VALIDITY

Internal Validity. The two major threats to internal validity are the quality of the surveys and logs. Surveys were filled by developers in the midst of their activities, sometimes under tight time constraints. As a result, we may have missed responses at the most pressing times, may have received incomplete or not well thought out responses, and the overall number and quality of the responses varied across developers. We tried to mitigate this risk by limiting the number and frequency of surveys per developer, and by including developers some of the authors knew to be responsive to this type of request even when under pressure.

It is also possible that there is noise in the logs we analyzed from spurious requests. To combat this, we were careful in spotting problematic patterns like duplicated records or requests that seem suspicious, and removing them from the data set. For example, we encountered some requests that happened too often because they were caused automatically by tools not developers. Although we attempted to understand and control for the variety of situations in which a search can be performed with the code search tool, the logging mechanism is complex. It is possible that we missed search events, or mistook an event as performing a search from the logs (e.g. following a link to search results). To mitigate this, we validated our assumptions about how the logs were constructed by running test searches and examining the generated logs. We also iterated on our methodology based on discussions with members of the code search team. Since the logging and survey collection mechanisms were not synced, we had to link the surveys to search sessions using timestamps. It is possible that in some cases our bindings

Table 7: Mapping Categories for Search (Table 1) to search whole-session patterns (Table 4) plus NS for "sessions" that contain no search event (e.g. browsing directories or reviewing prior results). The cells contain the number of responses for the row and column, or the % using as denominator of the row (%)

Example Code Needed (How)	S+C+	%	(S1|(S+C+)){2,}	%	S+	%	(S+1)+	%	S+CS+	%	NS	%	^C	%
API consumer needs help	21	38	12	22	8	15	2	4	5	9	5	9	2	4
Discover library for task	2	18	0	0	8	73	1	9	0	0	0	0	0	0
Example to build off	2	25	1	13	2	25	1	13	2	25	0	0	0	0
How to do something	2	29	3	43	1	14	0	0	0	0	0	0	1	14
Exploring Code (What)	S+C+	%	(S1|(S+C+)){2,}	%	S+	%	(S+1)+	%	S+CS+	%	NS	%	^C	%
Check implementation details	16	32	8	16	1	2	10	20	9	18	4	8	2	4
Browsing	2	20	1	10	3	30	0	0	0	0	2	20	2	20
Checking common style	0	0	1	33	1	33	1	33	0	0	0	0	0	0
Name completion	0	0	1	50	1	50	0	0	0	0	0	0	0	0
Code Localization (Where)	S+C+	%	(S1|(S+C+)){2,}	%	S+	%	(S+1)+	%	S+CS+	%	NS	%	^C	%
Reachability	4	20	6	30	6	30	0	0	3	15	0	0	1	5
Showing to someone else	3	30	2	20	1	10	3	30	1	10	0	0	0	0
Location in the depot	3	33	3	33	0	0	1	11	1	11	0	0	1	11
Determine Impact (Why)	S+C+	%	(S1|(S+C+)){2,}	%	S+	%	(S+1)+	%	S+CS+	%	NS	%	^C	%
Why is something failing	11	44	2	8	2	8	3	12	1	4	3	12	3	12
Understanding dependencies	4	33	3	25	0	0	2	17	1	8	0	0	2	17
Side effects of proposed change	3	75	1	25	0	0	0	0	0	0	0	0	0	0
Metadata (Who and When)	S+C+	%	(S1|(S+C+)){2,}	%	S+	%	(S+1)+	%	S+CS+	%	NS	%	^C	%
Trace code history	5	38	3	23	1	8	2	15	2	15	0	0	0	0
Responsibility	0	0	0	0	3	33	3	33	0	0	3	33	0	0

are not correct, e.g. if a developer did not fill out the survey immediately, as was instructed.

Construct validity. There are many aspects of the search activity that neither the logs nor the survey captured. In particular, we did not capture whether the search was successful, that is, whether the developer obtained the information they needed. This was intentional as to reduce the intrusion to the developers, but it is clearly something that future studies must take into consideration. Using the logs to estimate search success is tricky, and future work will evaluate how well result clicks represent query success.

We identify potential cases of query reformulation using sequential searches with no clicks and Levenshtein's distance at the word-level, which may not be representative of query reformulation or actual query edit distances.

External Validity. Google has a unique software development process, and the way Google developers interact with code search may not match the way external developers would. It is possible that the reasons for using the code search tools, types of queries, or the way those queries are constructed, may be different. Furthermore, although this is the first study to combine surveys with log analysis, it only included 27 developers that may not even represent the Google population of developers.

7. CONCLUSIONS AND FUTURE WORK

Based on surveys and logs of programmer behavior, we can learn about how and why code search is done, what a typical code search session looks like in duration and number of queries and clicks, and how patterns of behavior correlate with a developer's goals. We learn that code search is integrated in many development activities, from helping someone with a task to reviewing code, from finding an example to finding who was responsible for a change. We also learn that a search session is generally just one to two minutes in length and involves just one to two queries and one to two file clicks. While we have gleaned lots of interesting information from the search logs and surveys, there are still many unanswered questions that require further analysis.

In this paper, we did not focus on questions related to search quality at all. We defined query reformulation as two successive searches with no clicks in-between;. future studies will evaluate if this was an appropriate measure. When no results were clicked, it is unclear if this means the results page was sufficient to answer the question or if the query needs reformulating. When a result was clicked, we did not investigate the ranking of that result.

This study also did not focus on how the results are used, which could inform search tool design and workflow. We did attempt to gain any insight into the types of questions developers were not able to answer with code search. A deeper look into programmer behavior, likely involving direct observation, is needed to more fully understand the limitations of current search technology.

Distinct segments of the developer population may interact with code search differently. Some of the participants were software engineers and others were release engineers or test engineers. Each of these groups may have different search patterns. Future studies should include a large number of participants in each role to draw comparisons across groups. Different search patterns could emerge depending on developer experience. Future studies should control for this to see if less experienced developers search more, use longer query strings, or do other search activities differently when compared to developers with more experience.

Some of the results may be affected by the fact that the Google repository follows a very well defined and practiced set of company standards (e.g., directory structure and naming conventions) that may affect how people search and how good the results are. Reproducing the study in other companies would reveal the generality of these results.

Acknowledgements

This work is supported in part by NSF SHF-1218265, SHF-EAGER-1446932, the Harpole-Pentair endowment at Iowa State University, and a Google Faculty Award. Special thanks to Lars Clausen and Ben St. John at Google for their help.

8. REFERENCES

[1] S. Bajracharya, T. Ngo, E. Linstead, Y. Dou, P. Rigor, P. Baldi, and C. Lopes. 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*, OOPSLA '06, pages 681–682, New York, NY, USA, 2006. ACM.

[2] J. Brandt, M. Dontcheva, M. Weskamp, and S. R. Klemmer. Example-centric programming: Integrating web search into the development environment. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pages 513–522, New York, NY, USA, 2010. ACM.

[3] J. Brandt, P. J. Guo, J. Lewenstein, M. Dontcheva, and S. R. Klemmer. Two studies of opportunistic programming: interleaving web foraging, learning, and writing code. In *Proceedings of the 27th international conference on Human factors in computing systems*, CHI '09, pages 1589–1598, New York, NY, USA, 2009. ACM.

[4] Google Code Search for Chromium. http://cs.chromium.org/, March 2014.

[5] CodePlex. https://www.codeplex.com/, March 2014.

[6] Github: Build software better, together. http://https://github.com/, March 2014.

[7] Google Blog. Shutting down code search. Available from `http://googleblog.blogspot.com/2011/10/fall-sweep.html`, 2011.

[8] Google Code. https://code.google.com/, March 2014.

[9] M. Grechanik, C. Fu, Q. Xie, C. McMillan, D. Poshyvanyk, and C. Cumby. Exemplar: Executable examples archive. In *International Conference on Software Engineering*, pages 259–262, 2010.

[10] R. Hoffmann, J. Fogarty, and D. S. Weld. Assieme: Finding and leveraging implicit references in a web search interface for programmers. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*, UIST '07, pages 13–22, New York, NY, USA, 2007. ACM.

[11] W. Hudson. Card sorting. In M. Soegaard and R. Dam, editors, *Encyclopedia of Human-Computer Interaction, 2nd Edition*. The Interaction Design Foundation, 2013.

[12] A. J. Ko, B. A. Myers, M. J. Coblenz, and H. H. Aung. An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. *IEEE Trans. Softw. Eng.*, 32(12):971–987, Dec. 2006.

[13] Koders, August 2012.

[14] Krugle. http://opensearch.krugle.org/projects/, March 2014.

[15] J. Lawrance, R. Bellamy, M. Burnett, and K. Rector. Using information scent to model the dynamic foraging behavior of programmers in maintenance tasks. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '08, pages 1323–1332, New York, NY, USA, 2008. ACM.

[16] O. A. L. Lemos, S. K. Bajracharya, J. Ossher, R. S. Morla, P. C. Masiero, P. Baldi, and C. V. Lopes. Codegenie: Using test-cases to search and reuse source code. In *Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering*, ASE '07, pages 525–526, New York, NY, USA, 2007. ACM.

[17] H. Li, Z. Xing, X. Peng, and W. Zhao. What help do developers seek, when and how? In *Reverse Engineering (WCRE), 2013 20th Working Conference on*, pages 142–151, Oct 2013.

[18] E. Linstead, S. Bajracharya, T. Ngo, P. Rigor, C. Lopes, and P. Baldi. Sourcerer: mining and searching internet-scale software repositories. *Data Mining and Knowledge Discovery*, 18(2):300–336, 2009.

[19] A. Marcus, A. Sergeyev, V. Rajlich, and J. Maletic. An information retrieval approach to concept location in source code. In *Reverse Engineering, 2004. Proceedings. 11th Working Conference on*, pages 214–223, Nov 2004.

[20] C. McMillan, M. Grechanik, D. Poshyvanyk, C. Fu, and Q. Xie. Exemplar: A source code search engine for finding highly relevant applications. *Software Engineering, IEEE Transactions on*, 38(5):1069–1087, Sept 2012.

[21] Merobase. http://www.merobase.com, September 2013.

[22] Ohloh Code Search. http://code.ohloh.net/, September 2012.

[23] S. P. Reiss. Semantics-based code search. In *Proceedings of the International Conference on Software Engineering*, pages 243–253, 2009.

[24] C. Sadowski, J. van Gogh, C. Jaspan, E. Söderberg, and C. Winter. Tricorder: building a program analysis ecosystem. In *International Conference on Software Engineering (ICSE)*, 2015.

[25] Searchcode. searchcode, March 2014.

[26] S. Sim, M. Agarwala, and M. Umarji. A controlled experiment on the process used by developers during internet-scale code search. In S. E. Sim and R. E. Gallardo-Valencia, editors, *Finding Source Code on the Web for Remix and Reuse*, pages 53–77. Springer New York, 2013.

[27] S. Sim, C. Clarke, and R. Holt. Archetypal source code searches: a survey of software developers and maintainers. In *Program Comprehension, 1998. IWPC '98. Proceedings., 6th International Workshop on*, pages 180–187, Jun 1998.

[28] S. E. Sim, M. Umarji, S. Ratanotayanon, and C. V. Lopes. How well do search engines support code retrieval on the web? *ACM Transactions on Software Engineering Methodology*, 21(1):4:1–4:25, Dec. 2011.

[29] J. Singer, T. Lethbridge, N. Vinson, and N. Anquetil. An examination of software engineering work practices. In *Proceedings of the 1997 Conference of the Centre for Advanced Studies on Collaborative Research*, CASCON '97, pages 21–. IBM Press, 1997.

[30] Source Forge. http://sourceforge.net/, March 2014.

[31] K. T. Stolee, S. Elbaum, and D. Dobos. Solving the search for source code. *ACM Trans. Softw. Eng. Methodol.*, 23(3):26:1–26:45, June 2014.