

# Google-Driven Development: A Situated Study of Web Use in Programming

Elise Hein

HCI-E MSc Final Project Report 2017  
UCL Interaction Centre, University College London  
Supervisor: Nicolai Marquardt

## ABSTRACT

To support the culture of opportunistic programming—hacking, rapid iteration, and reuse of found code via copy-and-paste—researchers are increasingly exploring options of integrating online resources into the development workflow. Such explorations rely heavily on an understanding of how programmers search the Web. Most research, however, has been based on aggregate log analyses or laboratory experiments which fail to capture real-world context. In this study, we tracked the online searches of 18 programmers in a two-week instrumented panel while they were engaged in their normal day-to-day work activities. Based on first-hand commentary which participants added to a subset of their searches using a browser extension, we present seven classes of programming search goals and discuss specific search strategies. As a methodological contribution, we describe and evaluate the use of an instrumented panel coupled with experience sampling (IP-ES). Our results contribute to a theory of online resource usage in programming, and include guidelines for collecting annotated query logs in practice.

## Author Keywords

opportunistic programming; Web search for programming information; query log analysis; instrumented panel; diary study

## ACM Classification Keywords

H.5.2 Information Interfaces and Presentation: User Interfaces—*user-centred design, theory and methods*; H.5.m. Information Interfaces and Presentation (e.g. HCI): miscellaneous

## MSc Contribution Type

Empirical, Methodological

## MSC HCI-E FINAL PROJECT REPORT

*Project report submitted in part fulfilment of the requirements for the degree of Master of Science (Human-Computer Interaction with Ergonomics) in the Faculty of Brain Sciences, University College London, 2017.*

## NOTE BY THE UNIVERSITY

*This project report is submitted as an examination paper. No responsibility can be held by London University for the accuracy or completeness of the material therein.*

## CONTENTS

|            |   |           |
|------------|---|-----------|
| <b>1</b>   | <b>Introduction</b>                                 | <b>2</b>  |
| <b>2</b>   | <b>Related Work</b>                                 | <b>2</b>  |
| 2.1        | Web Use among Software Engineers . . . . .          | 2         |
| 2.2        | Investigating Online Search Behaviour . . . . .     | 5         |
| <b>3</b>   | <b>Method</b>                                       | <b>7</b>  |
| 3.1        | The Logging and Annotation Software . . . . .       | 8         |
| 3.2        | Mitigating Privacy Concerns . . . . .               | 8         |
| 3.3        | Conducting the Study . . . . .                      | 11        |
| 3.4        | Analysing the Data . . . . .                        | 14        |
| <b>4</b>   | <b>Findings: Developers' Online Search Patterns</b> | <b>16</b> |
| 4.1        | The Data Corpus . . . . .                           | 16        |
| 4.2        | Search Goals . . . . .                              | 16        |
| 4.3        | Search Strategies . . . . .                         | 19        |
| <b>5</b>   | <b>Method Evaluation</b>                            | <b>20</b> |
| 5.1        | Participation Rates and Data Volume . . . . .       | 20        |
| 5.2        | Participant Perspective . . . . .                   | 20        |
| 5.3        | Investigator Perspective . . . . .                  | 23        |
| <b>6</b>   | <b>Conclusion</b>                                   | <b>25</b> |
| Appendix A | Participant Recruitment Survey . . . . .            | 29        |
| Appendix B | Enrolment Survey . . . . .                          | 29        |
| Appendix C | Follow-up Survey Example . . . . .                  | 31        |
| Appendix D | Exit Interview Protocol . . . . .                   | 32        |
| Appendix E | Participant Correspondence . . . . .                | 33        |
| Appendix F | Information Sheet . . . . .                         | 37        |
| Appendix G | Informed Consent Form . . . . .                     | 39        |
| Appendix H | Browser Plugin Welcome Page . . . . .               | 40        |
| Appendix I | Exit Interview Themes . . . . .                     | 42        |
| Appendix J | Batches per Participant . . . . .                   | 43        |

## 1. INTRODUCTION

In the past decades, a set of alternative practices in software engineering has been widely described in the literature, collectively known as opportunistic programming. An approach that facilitates rapid prototyping, experimentation, and discovery [7], it has been viewed mostly in the context of end-user software engineering—programming by non-professionals [7, 10, 19, 23]—and contrasted with the more traditional *systematic* programming [43]. Recently, however, with start-ups and established companies alike beginning to embrace agile practices [2, 26], and the ranks of non-professional hackers and tinkerers growing [23], we can say that opportunistic practices are no longer as much about untrained end-users as they are about programming in the age of the Internet [19].

Among the five characteristics of opportunistic programming—rapid iteration, impermanent code, unique debugging challenges, building from scratch, and copy-and-pasting found code [7]—two in particular are largely facilitated by Web search engines. Building from scratch using high-level tools rather than modifying existing systems requires the ability to find tutorials and other learning resources, and copy-and-pasting found code requires the ability to find snippets relevant to specific APIs and assess their suitability. With a range of resources enjoying wide availability on the Web, online search is so pervasive in the workflows of today’s programmers that it is not uncommon for developers to occasionally doubt their programming ability, believing instead that their talent lies in skilled querying. Brandt et al. [6] (who’s work also serves as the main inspiration for this study) aptly note that nowadays programming is becoming less about knowing how to do something and more about asking the right questions.

Understanding *how* these questions are asked online can support programmers’ search practices through better designed tools, both in the browser and in the IDE (Integrated Development Environment) [6, 28]. Though a range of search engines and IDE extensions have been described in the literature (e.g., [5, 18, 20, 43]), industry adoption has been limited. One area to consider when overcoming these adoption barriers is research methodology.

Though field studies in HCI are today applied in a range of areas, the empirical study of programmers (ESP) is one of the longest-standing research concentrations within HCI [23]. Nonetheless, a lack of tool adoption by professional developers is a concern recognised by the ESP community, and one reason put forth is that the majority of studies is conducted in the lab, and tool design is not rooted in what software engineers actually *do* in their daily practice [14, 28, 34, 38, 41, 45]. Indeed, a large portion of research on how programmers search the Web is based on observations of artificial tasks or analyses of aggregate search query logs that provide little context.

One approach to keep research grounded in real practice is to apply *in situ* methodologies, many of which can be appropriated to the field of software engineering; for comprehensive reviews and comparisons, see, for example, [13, 29, 38]. Using a balanced mix of methods and appropriate application, field studies can reveal not only the *hows* and the *whats* of a given behaviour, but also the *whys*. In the case of programmer search

behaviour it is precisely the *whys* that house an abundance of implications for tool design. So, despite a significant body of research that has already contributed to theories of how programmers consume online resources, an opportunity exists to improve our understanding of these practices in the wild.

Motivated by expanding upon the findings of past research, and by developing a longitudinal *in situ* method appropriate to the field of software engineering, the aim of this study is to explore programmers’ online search practices during their day-to-day tasks. An *in situ* method will allow us to focus on programmers who are familiar with their codebase and undertake a variety of programming tasks—something that is difficult to simulate in a lab. Our research questions can thus be put simply:

1. What do professional programmers working on familiar projects search for on the Web?
2. What search strategies do they use in this process?

In this report, we present the results of a longitudinal study where 18 programmers anonymously shared their Web searches for two weeks. To capture contextual factors, participants also provided first-hand commentary for a subset of their searches. Discussed in Chapter 4, our findings contribute empirical evidence to the general theory of online resource consumption among programmers.

As an additional contribution, Chapter 3 presents the design and implementation details of our data collection apparatus: a browser extension that tracks participants’ Web searches and prompts them for timely commentary. With the methodological review given in Chapter 5, we hope to motivate increased use of similar *in situ* approaches in future empirical software engineering studies.

We begin, however, with a review of related work into programmers’ information needs and methods of investigating online search behaviour.

## 2. RELATED WORK

This research builds on two bodies of related work: (1) how software engineers use the Web, and (2) how online search behaviour is investigated.

### 2.1 Web Use among Software Engineers

#### 2.1.1 Information needs

To begin from a high level, the learning barriers described by Ko et al. [25] provide a blanket framework for categorising programmers’ information needs. A total of six barriers are identified—design, selection, coordination, use, understanding, and information—each addressing a different level of difficulty faced by programmers (see Figure 2.1 for details). Of the six, questions that arise at the barriers of understanding and information have posed long-standing research interests as they concern program comprehension, the fundamental cognitive activity software engineers engage in. Sillito et al. [39], for example, identified four question areas that represent barriers to understanding: finding initial focus points (e.g., *Where is the text in this error message defined?*), building on those points (e.g., *What data is being modified in this method?*),

understanding program subgraphs (e.g., *How is this UI error implemented?*), and understanding groups of subgraphs (e.g., *What will be the total impact of this change?*). On a more granular level, LaToza and Myers [27] listed 94 distinct questions, primarily also addressing understanding and information barriers; they include, among others, *What is the intent of this code? Is it possible to refactor this? How did this ever work?*

When we consider the context of a team and a project, questions external to the codebase also become relevant. Fritz and Murphy [15] identified 78 such questions, including *Why did this code change? What has changed between two builds? Has progress been made on blockers? Who broke the build?* Though mapping questions of this type onto any learning barrier would be contentious, they are important in recognising that programmers must locate and consolidate information of diverse types and from an array of sources.

Questions relating to program comprehension and teamwork, such as those exemplified above, can usually be answered by *examining* code and other artefacts. They also seldom yield to search mechanisms, bar the use of IDE-specific code search, `grep`, or version control logs. To overcome barriers of design, selection, coordination, and use, which represent *non-examinable* information needs, programmers often turn to the Web—it is this domain that the current study explores. It has been reported that programmers spend on average 19% of development time on the Web, spanning across many distinct search sessions [6]. Several studies have investigated how these search sessions unfold.

Brandt et al. [6], observing programmers building an online chat room in the lab, found that Web searches express three main goals: just-in-time learning of new skills and approaches, clarifications of existing knowledge, and reminders for details not worth remembering. Analysing query logs from Adobe’s Developer Network, they also linked the structure of a given query with its goal—the use of natural language signals a learning motivation, while the use of code terms indicates that a search for clarifications or reminders is more likely.

In addition to categorising Web searches by goal, dimensions such as topic area and resource type have been used. In their analysis of 15 million Windows Live Search queries, Hoffmann et al. [20] distinguished between such popular topics as APIs<sup>1</sup>, troubleshooting, implementations, and development tools. In a longitudinal field study, Goldman and Miller [16] developed a more generic categorisation: code elements, code ideas, sites, tools, and concepts. In an observational study where students worked on small projects, Stylos and Myers [43] listed several types of Web resources that were made use of, such as tutorials, documentation pages, and articles. Finally, though search behaviour was not the focus of their study, Hartmann et al. [18] found that programmers’ searches are not necessarily about programming, but may be motivated by an attempt to understand an application domain in general (e.g., colour theory or musical scales).

<sup>1</sup>Unless otherwise stated, mentions of APIs in the current report refer not to public service APIs (e.g., Twitter), but to programming interfaces in general, including method names and language features.

Related studies have also analysed the corpus of questions posted on popular programming Q&A websites such as Stack Overflow<sup>2</sup> [31, 44]. While the link between Web search and Q&A websites is contentious, it is reasonable to assume that questions are often posted as a result of failed Web searches (indeed, Stack Overflow was designed to be used such that Google is the UI [44]). According to Treude et al. [44], the most popular questions seek instruction (*How to crop an image by 160 degrees from centre in asp.net?*), explanations for unexpected behaviour (*iphone—Coremotion acceleration always zero*), or information on configuring development environments (*How to use windows emacs as svn client?*). This is aligned with other reports on Web search usage [6, 20, 43].

As a summary of programmers’ information needs, Figure 2.1 maps the search topics we have discussed onto six learning barriers that programmers face [25].

### 2.1.2 Comparisons with the general public

The literature also describes online search practices of the general public. As the scope of topics in this case is infinitely large, universal classifications are difficult. Rose and Levinson [37] propose a resource-navigation-information triotomy, whereby people search online to either obtain a resource, navigate to a known website, or learn by viewing an unknown website. Using this classification, they note that informational searches (not to be confused with the *barrier* of information) account for 60% of all sessions. Programming searches, however, are—by definition—mostly informational.

Differences also appear in query refinement practices, whereby users rephrase their search term to retrieve higher-quality results. According to Silverstein et al. [40], 23% of search sessions by the general public include some type of query refinement; in comparison, little of this behaviour has been observed in programmers [6] (the least when engaged in search for clarification or reminding). This could be attributed to the skill of the participants, the sophistication of search engines, or programming as a domain being especially suited for search [6] (indeed, search engines were originally optimised for informational queries).

### 2.1.3 Tooling to support search

Studies on programmers’ information needs make a valuable contribution to the design of tooling to support search. Currently, tools that attempt to bridge the gap between the browser and the IDE are seldom encountered [19]—though many have been presented in the literature, they are not in wide use in industry. Some, such as Blueprint [5], Codelets [32] and Seahawk [33], integrate search directly into the IDE, while others like Assieme [20] and Mica [43] provide an optimised search engine. The abstract domain of understanding rationale and context in a piece of code is also tackled—Codetrail [16] and HyperSource [18] generate a kind of meta-level documentation by linking the programmer’s historic browsing activities back to relevant files and snippets in the codebase.

We hope that the current research will help to refine future search tool design to better suit industry programmers’ work practices.

<sup>2</sup><http://stackoverflow.com>

LEARNING BARRIERS Ko et al. [25]

| Design   | Selection   | Coordination  | Use  | Understanding  | Information   |
|--|---|---|--|--|---|
| I don't know what I want the computer to do...   | I think I know what the computer should do, but I don't know what to use... | I think I know what things to use, but I don't know how to integrate them...        | I think I know what to use, but I don't know how to use it...  | I thought I understood this, but it didn't do what I expected...   | I think I know why it didn't do what I expected, but I don't know how to check...                             |
| <b>Novice</b><br>e.g., [Oracle PL/SQL performance tuning crash courses]                            | <b>Conceptual</b><br>e.g., [concept of XML sitemaps]                        | <b>How-to</b><br>e.g., [How to crop an image by 160 degrees in asp.net?]            | <b>Decision help</b><br>e.g., [Should a business object know about its corresponding contract object?] | <b>Discrepancy &amp; error</b><br>e.g., [iPhone - why is Coremotion acceleration always zero?], [C# Obscure error: file " could not be refactored] | Questions related to environment, non-functional questions, not related to programming, or asking for review. |
| <b>Concepts</b><br>e.g., [fibonacci heap]  | <b>Conceptual</b><br>e.g., [concept of XML sitemaps]                        | <b>How-to</b><br>e.g., [How to crop an image by 160 degrees in asp.net?]            | <b>Decision help</b><br>e.g., [Should a business object know about its corresponding contract object?] | <b>Discrepancy &amp; error</b><br>e.g., [iPhone - why is Coremotion acceleration always zero?], [C# Obscure error: file " could not be refactored] | Questions related to environment, non-functional questions, not related to programming, or asking for review. |
| <b>Identifying APIs</b>  | <b>Concepts</b><br>e.g., [fibonacci heap]                                   | <b>Code ideas</b><br>Implementation-level information, e.g., [read from input.java] | <b>Code elements</b><br>e.g., [java.lang.math]   | <b>Discrepancy &amp; error</b><br>e.g., [iPhone - why is Coremotion acceleration always zero?], [C# Obscure error: file " could not be refactored] | Questions related to environment, non-functional questions, not related to programming, or asking for review. |
| <b>Algorithms</b>  | <b>Concepts</b><br>e.g., [fibonacci heap]                                   | <b>Code ideas</b><br>Implementation-level information, e.g., [read from input.java] | <b>Code elements</b><br>e.g., [java.lang.math]   | <b>Discrepancy &amp; error</b><br>e.g., [iPhone - why is Coremotion acceleration always zero?], [C# Obscure error: file " could not be refactored] | Questions related to environment, non-functional questions, not related to programming, or asking for review. |
| <b>Tutorials</b>   | <b>Concepts</b><br>e.g., [fibonacci heap]                                   | <b>Code ideas</b><br>Implementation-level information, e.g., [read from input.java] | <b>Code elements</b><br>e.g., [java.lang.math]   | <b>Discrepancy &amp; error</b><br>e.g., [iPhone - why is Coremotion acceleration always zero?], [C# Obscure error: file " could not be refactored] | Questions related to environment, non-functional questions, not related to programming, or asking for review. |
| <b>Information on an API</b>   | <b>Concepts</b><br>e.g., [fibonacci heap]                                   | <b>Code ideas</b><br>Implementation-level information, e.g., [read from input.java] | <b>Code elements</b><br>e.g., [java.lang.math]   | <b>Discrepancy &amp; error</b><br>e.g., [iPhone - why is Coremotion acceleration always zero?], [C# Obscure error: file " could not be refactored] | Questions related to environment, non-functional questions, not related to programming, or asking for review. |
| <b>Implementation</b>  | <b>Concepts</b><br>e.g., [fibonacci heap]                                   | <b>Code ideas</b><br>Implementation-level information, e.g., [read from input.java] | <b>Code elements</b><br>e.g., [java.lang.math]   | <b>Discrepancy &amp; error</b><br>e.g., [iPhone - why is Coremotion acceleration always zero?], [C# Obscure error: file " could not be refactored] | Questions related to environment, non-functional questions, not related to programming, or asking for review. |
| <b>Learning</b><br>Learning by doing, often using tutorials  | <b>Concepts</b><br>e.g., [fibonacci heap]                                   | <b>Code ideas</b><br>Implementation-level information, e.g., [read from input.java] | <b>Code elements</b><br>e.g., [java.lang.math]   | <b>Discrepancy &amp; error</b><br>e.g., [iPhone - why is Coremotion acceleration always zero?], [C# Obscure error: file " could not be refactored] | Questions related to environment, non-functional questions, not related to programming, or asking for review. |
| <b>Clarification</b><br>User has high-level knowledge, but needs clarification                     | <b>Concepts</b><br>e.g., [fibonacci heap]                                   | <b>Code ideas</b><br>Implementation-level information, e.g., [read from input.java] | <b>Code elements</b><br>e.g., [java.lang.math]   | <b>Discrepancy &amp; error</b><br>e.g., [iPhone - why is Coremotion acceleration always zero?], [C# Obscure error: file " could not be refactored] | Questions related to environment, non-functional questions, not related to programming, or asking for review. |
| <b>Reminding</b><br>User knows what to do but needs syntactic/low-level reminders                  | <b>Concepts</b><br>e.g., [fibonacci heap]                                   | <b>Code ideas</b><br>Implementation-level information, e.g., [read from input.java] | <b>Code elements</b><br>e.g., [java.lang.math]   | <b>Discrepancy &amp; error</b><br>e.g., [iPhone - why is Coremotion acceleration always zero?], [C# Obscure error: file " could not be refactored] | Questions related to environment, non-functional questions, not related to programming, or asking for review. |
| <b>Troubleshooting</b><br>Syntax & semantics, development tools, conferences & magazines, licenses | <b>Concepts</b><br>e.g., [fibonacci heap]                                   | <b>Code ideas</b><br>Implementation-level information, e.g., [read from input.java] | <b>Code elements</b><br>e.g., [java.lang.math]   | <b>Discrepancy &amp; error</b><br>e.g., [iPhone - why is Coremotion acceleration always zero?], [C# Obscure error: file " could not be refactored] | Questions related to environment, non-functional questions, not related to programming, or asking for review. |

Brandt et al. [6]

Hoffman et al. [20]

Goldman and Miller [16]

Treude et al. [44]

Figure 2.1. A summary of studies on programmers' use of the Web based on a sample from the literature, mapped onto the learning barriers identified by Ko et al. [25] (note that mappings are notional only). Most Web use centres around overcoming barriers of selection, coordination and use; the Web is also used for resources that do not strictly map onto a learning barrier, such as syntax reminders, overhead management and navigation. Barriers of information, and to a lesser extent understanding, are more likely overcome by examining the code or communicating with teammates.

### 2.1.4 Changing use of the Web

Web use is an area of rapid change motivated by technological improvements, culture, as well as trends; thus, programmers' online search behaviour also exhibits subtle but decisive shifts over time [22, 37]. For example, from 1997 to 2001 a decrease in willingness to view more than one page of results was observed [37]—this could be explained by improved ranking algorithms that place higher-quality links closer to the top. Another change in behaviour that can be attributed to technological advancement is programmers' use of search interfaces as a translator between concepts in different languages [6, 17, 43]—a strategy largely facilitated by as-you-type results<sup>3</sup> and automatic suggestions. Query construction and refinement is also somewhat moulded by previous experience in what makes efficient search terms, prompting cultural shifts as new resources gather traction. For example, programmers today may get higher quality results when using queries that better suit Stack Overflow's question format [44].

Finally, though Google has been reported as programmers' search engine of choice in most studies, programming-specific services are also emerging: SymbolHound<sup>4</sup>, for example, retrieves higher-quality results for queries that include programming-specific terms, and Hoogle<sup>5</sup> is a similar service for Haskell-specific queries. In contrast, Google handles code symbols poorly at present [3] and would retrieve a collection of get-rich-quick sites in response to [make \$]<sup>6</sup>.

We hope that by adopting an in-the-wild approach, our study captures a range of current search use cases.

## 2.2 Investigating Online Search Behaviour

Several empirical methods are available for peering into users' online search patterns. Table 2.1 gives a brief account based on a sample of studies discussed above.

As illustrated, a common approach is to analyse query or transaction logs, which Jansen [21] defines as “an electronic record of interactions that have occurred during a searching episode between a Web search engine and users searching for information on that Web search engine”. In practice, this record can consist of search queries only, or search queries augmented with browsing history and other user interaction events. The current study adopts an approach where only browsing history is collected, and search queries are later extracted from URLs.

Depending on the nuances of the design, the method lies in varying distances from diary studies, ethnographic studies, and survey research [29, 38]. As such, it is applicable to an array of problem domains, from Web search personalisation to ranking algorithms [46]. But in the context of empirical software engineering, it has not been widely considered among other methods. For instance, though Lethbridge et al. [29]

mention tool log analysis in their comprehensive set of methods for studying software engineers, the tool considered is not necessarily a browser.

The benefit of retrospective log analysis lies in it being unobtrusive while generating significant amount of data from a sizeable number of users [21]. It also lends itself well for longitudinal studies, which are vital in discovering trends (e.g., [42]), as well as the full scope of topics that a search engine should support [17]. Finally, search and browsing logs are already automatically captured by most modern browsers, which simplifies the data gathering process [21].

On the other hand, some obvious weaknesses are also inherent in the method:

- User sessions are normally inferred from the IP associated with a transaction, but one computer may have multiple users [21].
- The log lacks user information such as demographic data [21].
- Query log analysis can be seen as one use case among the emerging applications of big data analysis. As such, an established notion of privacy for the use of query logs has yet to be defined [46]. Researchers can, of course, protect individuals' privacy by excluding certain metadata from an event log, but this is technically challenging to automate [46], and also imposes limits on the analysis [1, 46] (Figure 2.2).
- Individual events in the log are not a rich source of contextual and qualitative data. Namely, while a detailed trail of events is visible, it is difficult to gauge higher-level goals [6, 17, 21].

Note that this last point remains the subject of debate. Rose and Levinson [37], for instance, claim that by making use of *all* information attached to a search event—the query, the results returned, the results clicked on, and further search actions pursued—the goal of a given session may be accurately classified. However, the goal taxonomy for testing this claim was developed in the context of searches made by the general public, and whether it is granular enough to be applicable to programming queries remains open to question.

While the most common, log analysis is not the only method for studying search behaviour. Grimes et al. [17] compare aggregate query log analysis to two other approaches: observational studies (*in situ* or in the lab), where participants are observed as they perform tasks; and instrumented user panels, where users install logging software on their computers for some period of time. Both approaches account for the drawbacks inherent in aggregate log analyses by supplying richer contextual information, but in doing so they trade off scale, privacy and, to some degree, naturalness. Between the two, the instrumented panel evokes the more serious privacy concerns, as participants may be apprehensive of making their browsing data available over an extended period. As an approach similar to diary methods, it is also vulnerable to limitations such as memory biases and high participation burden [4]. On the other hand, data from instrumented panel are rich in context, as

<sup>3</sup>For example, Google Instant: <http://support.google.com/websearch/answer/186645>

<sup>4</sup><http://symbolhound.com>

<sup>5</sup><http://haskell.org/hoogle>

<sup>6</sup>The notation [query] is used to indicate a verbatim Web search query here and in all future cases.

| Study                                  | Method                                       | Details  | Scale  |
|--|--|--|--|
| <b>General public search behaviour</b> |  |  |  |
| Silverstein et al. [40]                | Query log analysis                           | Retrospective analysis of search queries to the AltaVista search engine over a period of six weeks   | ~285 million user sessions, 1 billion queries      |
| Spink et al. [42]                      | Query log analysis                           | Retrospective analysis of three sets of search queries to the Excite Web search engine collected in 1997, 1999 and 2001  | ~200 000 users, 1 million queries                  |
| Rose and Levinson [37]                 | Query log analysis                           | Retrospective analysis of search queries to the AltaVista search engine  | ~100–200 queries                                   |
| <b>Programmer search behaviour</b>     |  |  |  |
| Stylos and Myers [43]                  | Observation                                  | Observation of ongoing progress in Java student projects*  | 3 student participants                             |
| Hoffman et al. [20]                    | Transaction log analysis                     | Retrospective analysis of java-related queries and click-through data from the MSN search engine 339 search sessions   |  |
|  | Controlled experiment                        | A comparison of search performance using Google and Assieme by observing participants completing artificial search tasks   | 9 student participants                             |
| Goldman and Miller [16]**              | Instrumented user panel                      | Analysis of Eclipse events and Firefox page loads over the course of 1–3 weeks   | 4 participants; 646 development-related page loads |
|  | Instrumented user panel & demographic survey | Analysis of Eclipse and Firefox programmatic and user interface events over the course of 17 days on average   | 11 participants                                    |
| Brandt et al. [6]                      | Controlled experiment                        | An observation of participants' Web use while building an online chat room   | 20 student participants                            |
|  | Query log analysis                           | Retrospective analysis of search queries to Adobe's Developer Network search engine  | 24 293 users, 101 289 queries, 69 955 sessions     |
| Brandt et al. [5]                      | Controlled experiment                        | An observation of participants' search for example code while completing artificial coding tasks and using Blueprint (Adobe Community Help Search engine in the control group) | 20 professional programmers                        |

\* They also analysed screen recordings from a controlled experiment, but as the goal was not to study search behaviour, it is not included here.

\*\* Note that four of the participants in both studies were a part of the research group.

**Table 2.1. Empirical methods adopted in a sample of studies that investigate online search behaviour. Though several other studies reviewed in Section 2.1 used empirical approaches, studies whose main goal was not to investigate search behaviour (e.g., tool evaluations) have been omitted here for brevity.**

|                                 | Depth  | “Naturalness”                        | Flexibility                                   | Scale             | Turnaround           |
|---------------------------------|--|--------------------------------------|---|-------------------|----------------------|
| <b>Observational studies</b>    | Very detailed                                | Observed, tasks may be artificial    | Altered midstream as required                 | O(50) users       | ~1 month             |
| <b>Instrumented user panels</b> | Observes computer environment, multi-tasking | Natural, tasks may be edited by user | Hard to alter data collection                 | O(100) users      | ~2–4 weeks           |
| <b>Aggregate log analysis</b>   | Limited, no contextual information           | Completely natural                   | Easy to run experiments on Search Engine side | Millions of users | Real time to ~1 week |

**Table 2.2. A comparison of aggregate log analysis, instrumented user panels and controlled experiments, reproduced from Grimes et al. [17].**

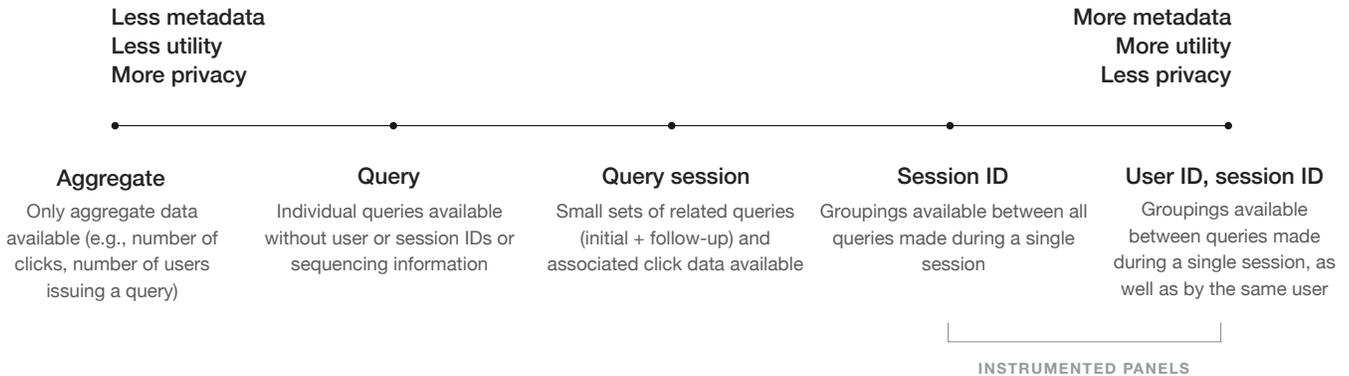


Figure 2.2. The privacy-utility spectrum of query log analysis, based on Xiong and Agichtein [46].

participants are known individuals and can provide additional details (refer back to Figure 2.2).

Table 2.2 summarises the three approaches on the dimensions of depth, naturalness, flexibility, scale, and turnaround of data. As the approach that combines the benefits of context provided by controlled experiments, the unobtrusiveness provided by log analysis, and the longitudinal nature of diary studies, the current study employs an instrumented user panel, which is especially suitable for capturing real-world context. For example, the artificial programming tasks prescribed in lab studies mostly involve building new features (refer back to Table 2.1), but programmers have been shown to engage in a variety of coding tasks during their daily work [28]. Among other studies in the literature investigating programmers’ search behaviour, that conducted by Goldman and Miller [16] represents the only other use of an instrumented panel to our knowledge.

### 2.2.1 Conducting query log analysis in practice

Few practical guides exist to conducting query and transaction log analyses in sufficient detail to replicate the method [21]. This presents a problem especially in the case of instrumented panels, where the researcher must collect data in real time from known participants (using a pre-existing dataset such as in [6, 20] would make combining it with contextual data from other sources difficult). While Jansen [21] implemented a logging tool to do just this, it is no longer available to the public. The commercial landscape is similarly lacking: despite an abundance of off-the-shelf logging tools for conducting user studies (e.g., Morae<sup>7</sup>), they are typically optimised for short-term use and would be unmanageable over longer periods. The researcher also has the option of using commercial time tracking software (e.g., RescueTime<sup>8</sup>, TimeDoctor<sup>9</sup>) by instructing users to manually export and share the data periodically. This presents three distinct disadvantages: (1) it is a threat to the inherently unobtrusive nature of query logging; (2) it requires finding creative ways of associating anonymous

<sup>7</sup><http://techsmith.com/morae.html>

<sup>8</sup><http://rescuetime.com>

<sup>9</sup><http://timedoctor.com>

logs with other data; and (3) researchers have no control over the format of the data, or which events are logged.

As such, a bespoke data collection tool has been developed for the current study. A detailed description of this is given in Chapter 3, where we also demonstrate how it overcomes some of the weaknesses inherent in instrumented user panels.

## 3. METHOD

This chapter outlines the data collection and analysis methods used in our study to explore development-related searches in the wild. Our research questions were:

1. What do professional programmers working on familiar projects search for on the Web?
2. What search strategies do they use in this process?

Based on the review of related work on programmers’ information needs and approaches to query log analysis discussed in Chapter 2, a set of criteria was established for the study:

**Real-world context.** To capture a diversity of tasks and search queries [17], logs should originate from programmers working on real projects following their own schedules and using routine search strategies.

**Unobtrusiveness.** To encourage participants to behave as naturally as possible [17], any impact on the programmer’s usual patterns of Web use should be minimised during data collection.

**Anonymity.** To address privacy concerns around making their browsing activity available [1, 46], participants’ browsing data should remain fully anonymous even from the investigators.

**First-hand commentary.** To address difficulties in understanding search goals [6, 17, 21], participants should annotate logs with contextual information.

We sought to meet these criteria by combining a longitudinal instrumented panel approach with the experience sampling element often found in diary studies [4, 36]; we refer to this

approach as the IP-ES (Instrumented Panel and Experience Sampling) method. Before outlining procedural details, this chapter describes the design of the software developed for conducting the study.

### 3.1 The Logging and Annotation Software

The tool presented here has two main features: the researcher’s ability to collect continuous browsing logs from a known participant for a set time period; and the participant’s ability to anonymously annotate a sample of the logs with contextual information. A suitable existing tool to achieve these means was not identified—a custom system was thus designed and built, integrating a commercial survey platform with the automated collection of browsing logs. Indeed, query log analysis is often complemented with traditional long-form questionnaires for participant data (e.g., [9, 35]); in this case, however, the survey platform was employed foremost as an experience sampling tool.

The system comprises of a secure database and server, a browser extension for the participants (the “instrument” in “instrumented panel”), and a data exploration interface for the investigator. TypeForm<sup>10</sup> was chosen as the survey platform due to its simple public-facing Web API and the option to supply hidden fields to surveys (e.g., participant ID)<sup>11</sup>. These components are illustrated in Figure 3.1 (p. 9).

Central to the system from the participants’ point of view is the browser extension. Upon installing the extension, a continuous background process is launched that sends one batch of browsing logs, extracted from the browser’s history, to the research database approximately once per 24 hours for two weeks, facilitating iterative and continuous data analysis. Though search patterns were the primary research interest, the extension collects *all* URLs from the browser’s history; this is due to the technical intricacy of log classification and the desire to capture a variety of (possibly lesser-known) search engines.

The extension also handles the experience sampling element of the study: for each batch of logs, participants are prompted to fill in follow-up surveys for search queries flagged by the investigator, providing contextual annotations. Further, the extension provides a link to a one-off enrolment questionnaire to be filled out in the beginning of the study. Its use lies in collecting wider demographic and contextual information about each participant, such as their experience and work practices as a programmer.

Figure 3.2 shows a screenshot of the plugin during a hypothetical study. On the investigators side, the system provides a secure data management interface for analysing incoming batches and flagging programming queries for follow-ups; this admin panel can be seen in Figures 3.3 and 3.4 (p. 10 and 11).

<sup>10</sup><http://typeform.com>

<sup>11</sup>TypeForm is also widely considered a modern, accessible survey platform, though we would point out its focus on expert computer users. Working with programmers, we assumed that a lack of experience with computers will not cause usability issues. Of course, issues caused by other factors are still a possibility.

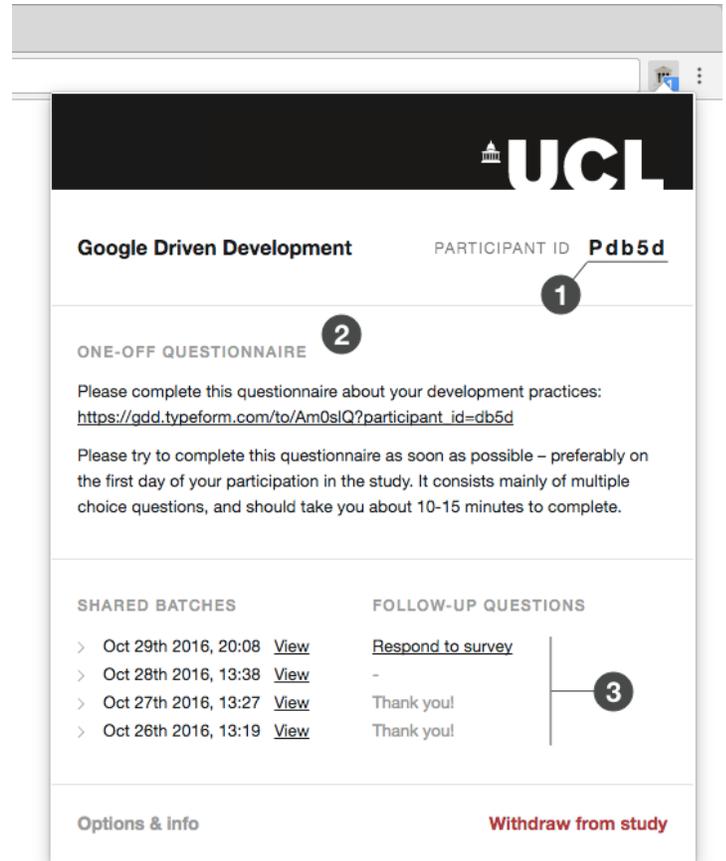
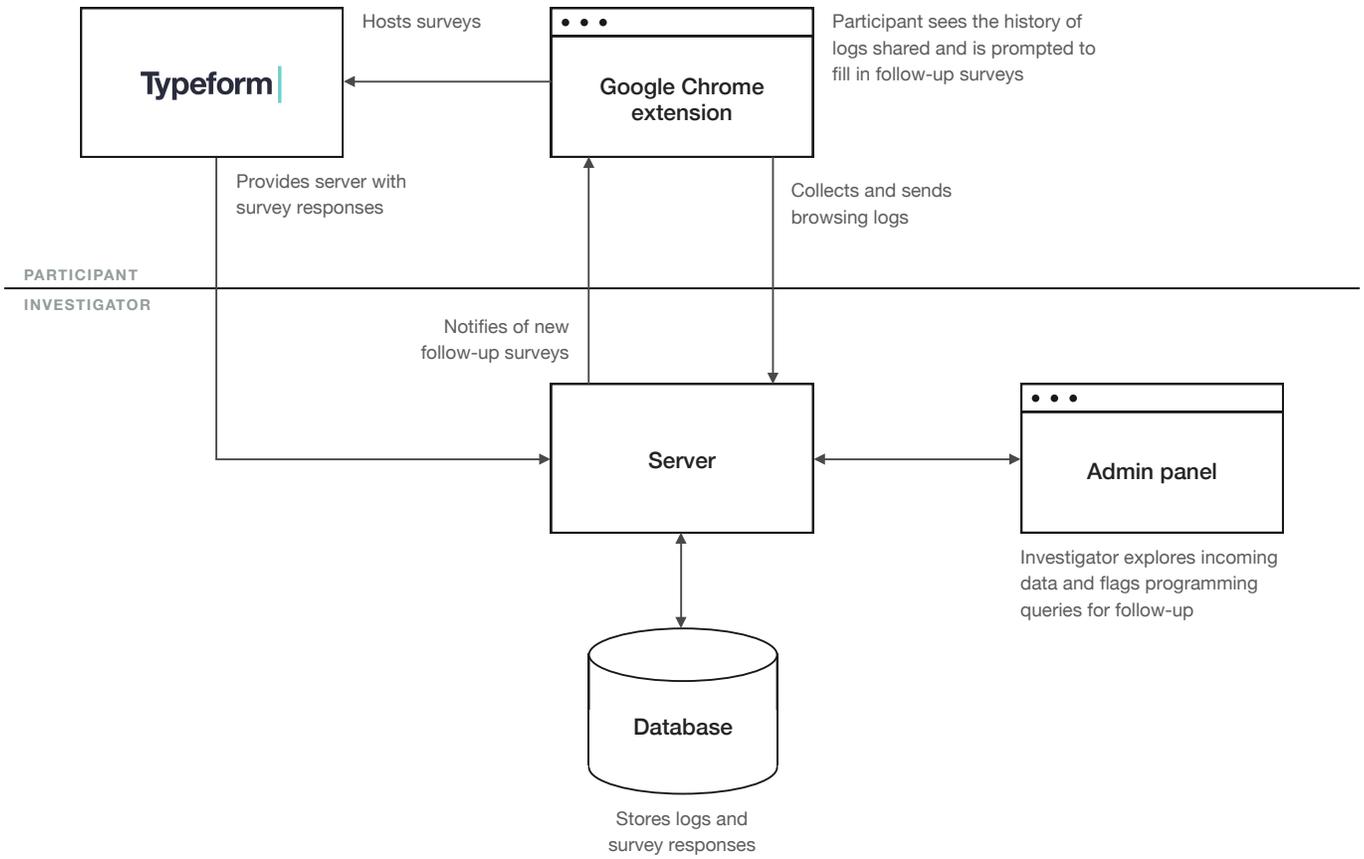


Figure 3.2. The browser extension installed by participants, as seen in the middle of a hypothetical study. The screenshot shows (1) a randomly generated participant ID used to associate survey responses to specific logs; (2) a link to the enrolment survey (this is highlighted in the beginning of the study and disappears automatically once the survey has been completed); and (3) a list of batches that have been shared along with links to any corresponding follow-up surveys, if available. The integration with TypeForm enables the extension to display a friendly “Thank you” message where follow-up surveys have been completed.

### 3.2 Mitigating Privacy Concerns

By adopting an approach that prioritises real-world context, the instrumented panel sacrifices a great deal of privacy [1, 17, 46]. It was therefore paramount to provide participants with options of reducing perceived invasiveness and increasing sense of control. The design of the system reflects this effort in the following features.

**Authentication tokens.** Though participants in an instrumented panel are personally recruited and therefore known individuals, the use of server-generated (but locally stored) participant IDs and authentication tokens provides a layer of anonymity that prevents their association to a given batch of logs. The participant ID is forwarded as a hidden field to surveys in lieu of names and emails, while the authentication token, invisible to the participant, is included in the HTTP header of each request made to the server, preventing malicious requests from third-party devices. Though this mechanism prevents identifying individuals directly, full anonymity cannot be guaranteed—falling under the



**Figure 3.1. The components in the IP-ES tool.**

### Incoming batches

[Classify logs](#) →

All participants
Incoming batches

| Nov 9th | PID                   | Logs | Follow-up status |
|---------|-----------------------|------|------------------|
| 15:46   | <a href="#">P8YjG</a> | 188  | ●                |
| 14:39   | <a href="#">P6U3J</a> | 73   | N/A              |
| 14:04   | <a href="#">P9qKK</a> | 104  | ●                |
| 13:44   | <a href="#">Pc67i</a> | 156  | ●                |
| 11:19   | <a href="#">PHQjR</a> | 78   | N/A              |
| 09:28   | <a href="#">PErDL</a> | 265  | ●                |
| 09:22   | <a href="#">Pvumj</a> | 65   | ●                |
| 07:39   | <a href="#">PGjSr</a> | 324  | ●                |
| 07:05   | <a href="#">PfvZr</a> | 166  | ●                |
| Nov 8th | PID                   | Logs | Follow-up status |
| 19:25   | <a href="#">Pesyd</a> | 245  | ●                |
| 18:47   | <a href="#">PsQBE</a> | 10   | N/A              |

#
LwNs7FRlsLcpBaX7

Uploaded
Nov 9th, 13:44

Logs
156

Follow-up status
●
[Verify survey](#)

FLAGGED FOR FOLLOW-UP

🔗 [SerializableResource - Google Search](#)
14:19

““ [SerializableResource](#)

*Wanted to know the structure of this class in Rails.*

BUILDING

PROGRAMMING QUERY

🔗 [moment - Google Search](#)
15:42

““ moment

*I wanted to look at the moment.js docs to find the formatting options for them.*

BUILDING
UNDERSTANDING

PROGRAMMING QUERY

🔗 [json parser - Google Search](#)
14:34

““ json parser

*I had a json from an API that I wanted to be parsed so I could interact with it properly*

TESTING

PROGRAMMING QUERY

Figure 3.3. The admin interface used by the investigator to explore incoming data and flag programming queries for follow-ups. Batches are grouped on the left by date, while all logs in the chosen batch is shown on the right. The screenshot given here illustrates a view of logs that have already been annotated by the participant; see Figure 3.4 for a view of raw incoming logs.

|   |                                     |
|---|-------------------------------------|
| “ Query for follow-up   | PROGRAMMING QUERY                   |
|  <b>aws query multiline text</b> <a href="#">Google Search</a> 20:08           | <input checked="" type="checkbox"/> |
| “ Query for follow-up   | PROGRAMMING QUERY                   |
|  <b>amazon ec2 - AWS CLI Command Line: How to u...</b> 20:08                   | <input type="checkbox"/>            |
| “ Query for follow-up   | PROGRAMMING QUERY                   |
|  <b>json array to multiline aws</b> <a href="#">Google Search</a> 20:13        | <input checked="" type="checkbox"/> |
| “ Query for follow-up   | PROGRAMMING QUERY                   |
|  <a href="#">GitHub</a> 20:13  | <input type="checkbox"/>            |
| “ Query for follow-up   | PROGRAMMING QUERY                   |
|  <b>output text aws</b> <a href="#">Search · output text aws</a> 20:13         | <input checked="" type="checkbox"/> |
| “ Query for follow-up   | PROGRAMMING QUERY                   |
|  <b>output text aws language:bash</b> <a href="#">Search · output...</a> 20:13 | <input checked="" type="checkbox"/> |
| “ Query for follow-up   | PROGRAMMING QUERY                   |
|  <b>ruby system call</b> <a href="#">Google Search</a> 20:18                 | <input checked="" type="checkbox"/> |
| “ Query for follow-up   | PROGRAMMING QUERY                   |
|  <a href="#">Getting output of system() calls in Ruby - Stack...</a> 20:18   | <input type="checkbox"/>            |

**Figure 3.4.** A view of a given participant’s raw logs on the admin panel. To reduce the manual labour required for filtering, the interface visually highlights logs that can be confidently considered online searches based on their URL pattern.

category of partial de-identification [46], it is vulnerable to data-linkage attacks whereby identities can be inferred indirectly (e.g., from metadata found in the logs).

**Timeframe.** The extension provides settings for configuring a timeframe during which the collection of browsing activity is allowed. For example, participants can specify a timeframe of 18:00–23:00 if they engage in most of their development activity in the evenings.

**Blacklisted URLs.** Participants can exclude specific URLs from shared batches. The system instructs participants to make use of regular expressions when adding items to the blacklist—as our user group consisted entirely of programmers, we were able to assume some familiarity with regular expressions, allowing for fine control over log filtering.

**Ability to review what has been shared.** Participants can download copies of batches for review (Figure 3.2).

In addition to the above, we encouraged participants to make use of private browsing (Incognito) mode for browsing that they wished to keep private.

### 3.3 Conducting the Study

This study was approved by the UCL Interaction Centre Ethics Chair, under the project ID Number UCLIC/1617/001/MSc Marquardt/Hein.

#### 3.3.1 Participants

18 participants were recruited (14 male, 4 female; average age of 32.1) for the study via a short recruitment survey posted on social media and mailing lists (Appendix A). (Particular effort was made to increase exposure to female programmers by advertising to multiple women-only engineering groups). The purposes of the survey were two-fold: (1) to ensure a significant amount of time spent programming each day (in order to generate enough relevant logs), and (2) to ensure Google Chrome as a browser of choice (the only browser at the time compatible with the instrument). A prize draw between all participants for two £50 Amazon gift vouchers was offered as remuneration.

Participants represented a range of nationalities, but nearly half resided in the UK. Most had between 3–10 years of professional programming experience, and were engaged in a professional software project throughout the duration of the study, working mostly alone or in small teams of up to four people. Rating their familiarity with the codebases and technologies used in their ongoing projects on a 7-point Likert scale, an average familiarity of 5.28 (SD = 1.5) and 5.83 (SD = 1.1) was reported, respectively. Our participants can thus be said to represent industry software engineers who are past the novelty effect of a new project, but still work in teams and may not intimately know all areas of the codebase or technologies used. See Table 3.1 for full participant profiles.

#### 3.3.2 Procedure

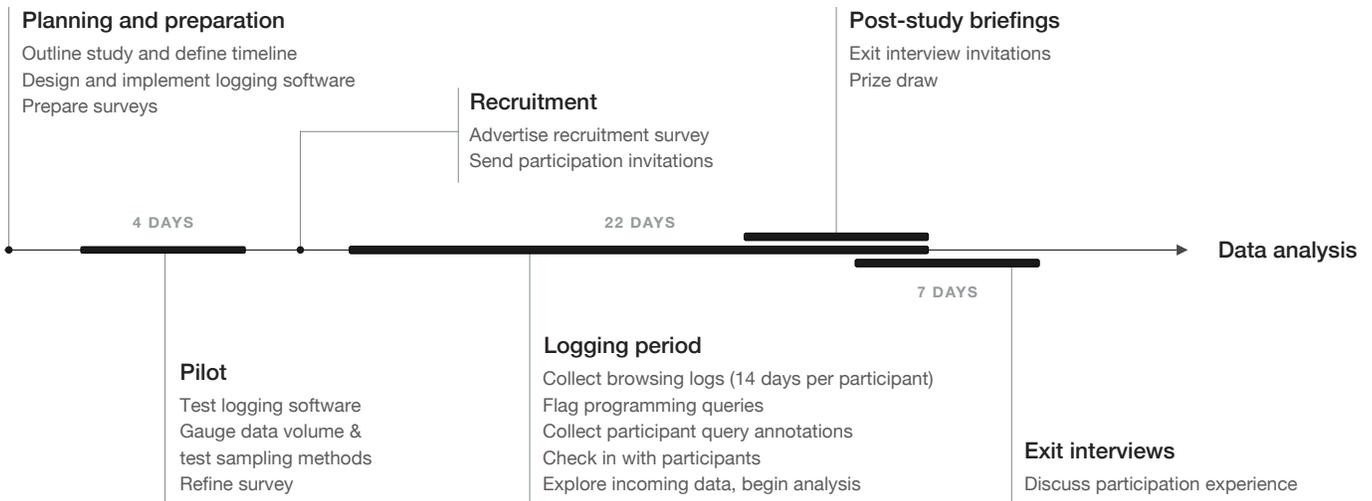
Figure 3.5 illustrates the main phases of the study: (1) piloting of the instrument and the surveys, (2) collection of programming-related searches and follow-up annotations, and (3) exit interviews.

*Preparations.* A pilot study was conducted with a single participant (27-year-old male involved in an intensive programming project) over a period of five days to test the instrument and gauge the volume of data that would be generated. The pilot allowed the admin panel to be equipped with suitable analysis and sampling tools and iterate on appropriate wording in the survey questions. Further, an assessment was made of the demands placed on participants when completing follow-up surveys, which were modified to fall within a completion time window of less than five minutes [4].

For the main study, participants who responded to the recruitment survey were invited to take part via an email detailing the purpose of the research (Appendix E lists all correspondence with participants), and including digital copies of the information sheet and consent form (Appendices F and G).

|       | Gender | Age | Home region   | Experience  |         |              | Current project | Team size       |          |
|-------|--------|-----|---------------|-------------|---------|--------------|-----------------|-----------------|----------|
|       |        |     |               | Total years | Student | Professional |                 |                 | Hobbyist |
| DSP1  | M      | 22  | Europe        | 1           | •       |              | •               | Student project | 2–4      |
| DSP2  | M      | 38  | South America | 1           |         | •            |                 | Professional    | 1        |
| DSP3  | F      | 25  | Europe        | 1–2         | •       | •            | •               | Professional    | 2–4      |
| DSP4  | M      | 25  | North America | 3–5         | •       | •            | •               | Professional    | 2–4      |
| DSP5  | M      | 26  | Europe        | 3–5         |         | •            | •               | Professional    | 11–20    |
| DSP6  | M      | 27  | Europe        | 3–5         | •       | •            |                 | Personal        | 1        |
| DSP7  | F      | 30  | North America | 3–5         |         | •            | •               | Professional    | 20+      |
| DSP8  | M      | 24  | Europe        | 6–10        | •       | •            | •               | Professional    | 2–4      |
| DSP9  | M      | 27  | Europe        | 6–10        |         | •            |                 | Professional    | 2–4      |
| DSP10 | M      | 31  | Asia          | 6–10        | •       | •            |                 | Professional    | 1        |
| DSP11 | M      | 33  | Europe        | 6–10        | •       | •            | •               | Personal        | 2–4      |
| DSP12 | F      | 35  | Asia          | 6–10        | •       | •            |                 | Professional    | 1        |
| DSP13 | F      | 35  | Europe        | 11–20       | •       | •            | •               | Professional    | 11–20    |
| DSP14 | M      | 36  | Europe        | 11–20       | •       | •            | •               | Professional    | 5–10     |
| DSP15 | M      | 38  | Europe        | 11–20       |         | •            |                 | Professional    | 5–10     |
| DSP16 | M      | 34  | North America | 20+         | •       | •            | •               | Professional    | 2–4      |
| DSP17 | M      | 46  | Europe        | 20+         |         | •            | •               | Professional    | 1        |
| DSP18 | M      | 46  | Europe        | 20+         |         | •            | •               | Professional    | 5–10     |

**Table 3.1. Participant profiles.** “DSP” refers to Diary Study Participant, representing instrumented panel users. “Diary Study” was chosen so as not to confuse “IPP” (Instrumented Panel Participant) with “IP” (Interview Participant).



**Figure 3.5. Timeline and main phases of the study.** Note that each individual participant’s data collection lasted for 14 days, but participants began the study at different times, so logs were collected for 22 days in total.

---

**On this day, what were you doing when you searched for...?**

---

**Designing**

You were analysing a new problem and mapping out the broad flow of code for solving it (e.g., exploratory research for a new feature, API design).

**Building**

You were producing code for new program behaviour (e.g., adding a new feature) and getting it into a compatible state.

**Editing**

You were editing existing code and returning it to a compatible state (e.g., fixing bugs, refactoring, or rearchitecting).

**Understanding**

You were determining information about code such as the inputs and outputs to methods, what the call stack looks like, why the code is doing what it is doing, or the rationale behind a design decision (e.g., investigating a bug, reviewing pull requests).

**Testing**

You were creating or running tests, or otherwise ensuring that code is behaving as expected (using a systematic approach outside of your main edit-debug cycle).

**Managing overhead**

You were engaged in activities related to the development environment, such as setting up repositories, build or deployment scripts, or managing dependencies.

**Other**

*Optionally specified by participant*

---

**Table 3.2.** The choice of programming-related activities offered to answer the question *What were you doing when you made this search?*

Participants who accepted the invitation were asked to return a signed consent form and to indicate a preferred start date for their participation.

On the first day of the study, participants received instructions for the set-up and use of the browser extension. Upon installation, a welcome page presented users with further details of how data would be collected for the following two weeks, and the configuration options described in Section 3.2 (Appendix H). It also prompted them to fill in the enrolment survey (Appendix B).

*Data collection.* For each batch of logs shared during the 14-day period, the researcher manually flagged three programming-related search queries for a follow-up; these were used by the system to automatically populate a TypeForm survey. Participants were prompted to fill in new follow-ups by a notification icon visible on the browser extension. 12 participants also opted to receive daily SMS survey reminders.

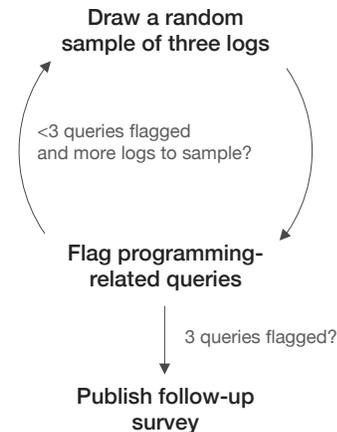
The survey asked participants to recall the following for each flagged query (paraphrased for brevity):

1. What was the trigger for this search? What problem were you trying to solve?
2. What were you doing when you made this search?

The first question required a short free-text response, and the second offered a choice between the most common programming-related activities identified by LaToza and My-

ers [28], definitions paraphrased in lay terms; these are listed in Table 3.2. The full survey with example searches is given in Appendix C.

*Query flagging.* To minimise the risk of biased retrospection and maximise recall accuracy in the follow-ups [4, 12], flagging search queries promptly was essential. To this end, the system sent notifications of new batches to the investigator and highlighted them in the admin panel. The three search queries were then flagged using an iterative sample-and-classify method built into the interface, illustrated in Figure 3.6.



**Figure 3.6.** The iterative sample-and-classify method used in the system to flag three random programming search queries from a batch of logs without manually classifying them all.

The sample-and-classify approach reduced the labour-intensity of manual filtering, ensuring also that a variety of queries was selected without bias. However, as the study progressed, we occasionally ruled against certain queries that emerged with the method to include a diverse set of searches and to avoid duplicates (for example, query refinements part of a single search session such as [mongo distinct date] and [mongodb distinct date from datetime]). In the end, a follow-up was only published if the batch contained at least three search queries from separate search sessions. To further aid participants in recalling the context of the flagged searches, each was accompanied in the survey by a timestamp indicating when the search was made.

*Exit interviews.* Following the data collection phase of the study, participants were thanked for their contribution via email and subsequently invited to take part in semi-structured exit interviews (note that participants were told that declining to be interviewed would not impact their participation in the prize draw). The invitation was accepted by nine participants (8 male, 1 female, average age of 29.3).

To preserve their anonymity, interviewees were not asked to elaborate on the searches they made during data collection. Instead, the interviews were used as a complementary method to evaluate the design of the study and participant perceptions towards notions of privacy, control, and awareness of their search patterns while being tracked (the interview protocol

is provided in Appendix D). All interviews were conducted in-person within at most four days after data collection period and lasted between 10–20 minutes.

### 3.4 Analysing the Data

Three distinct datasets were generated during the study:

1. the browsing logs gathered from participants during the two-week data collection period,
2. a set of annotated search queries within these logs, and
3. exit interview transcripts.

To identify programmers’ search motivations and strategies, we applied thematic analysis to the annotated search query log. The full log corpus was also subjected to basic data aggregation and statistical analysis, but its value lay primarily in its metadata, as participants left behind an electronic trail of their interactions with the browser plugin and surveys (a benefit of digital experience sampling [4]). Triangulated with themes drawn from exit interviews, this supported the evaluation of the IP-ES method (Chapter 5).

Responses to the enrolment survey formed a fourth dataset, but were not separately analysed. Instead, they were used to create participant profiles and cross-check themes emerging from the annotated query set (for example, to verify whether a search topic lay within a participants’ expertise).

#### 3.4.1 Analysis of the annotated query logs

*Data monitoring and preparation.* To facilitate iterative analysis (and to prevent the large volume of qualitative data from becoming unwieldy), incoming logs were continuously monitored and prepared for analysis.

For monitoring, a Metabase<sup>12</sup> dashboard was set up on the research database. This enabled us to keep track of data volume and surveys completed, but also to run SQL queries for ad hoc questions such as the range of search engines used.

On the admin panel, all logs were labelled as “programming query” or “other browsing”, and cases of query refinements were identified in the subset of annotated queries. Here, a sequence of queries with no gaps longer than five minutes was automatically identified as a search session [6, 40]. The number of query refinements in each identified session was then manually verified.

To allow iterative coding during data collection, an automated transfer was set up between TypeForm and Trello<sup>13</sup>. Trello is a Web-based interface normally used to manage tasks in a kanban workflow. Because of its comprehensive tagging, filtering and grouping features, however, the platform was employed as a coding tool. New TypeForm responses triggered the automatic creation of new Trello cards, each representing an annotated search query (seen in Figure 3.7). Meanwhile, this queue of cards was continuously processed by the researcher for coding and making note of cross-references. When new filtering needs emerged based on enrolment survey responses

(e.g., comparisons across level of experience), cards were tagged in bulk with ad hoc calls to Trello’s and TypeForm’s APIs.

The data monitoring and preparation workflow is illustrated in Figure 3.7.

*Open coding.* Annotated queries were coded using an inductive, bottom-up approach [8, 36] to signify

1. suspected search motivation;
2. discrete characteristics, ranging from the explicit (such as whether programming terms are present in the query) to the interpretive (such as whether the annotations indicate collaboration with fellow programmers).<sup>14</sup>

With reference to the first point, the approach is similar to that adopted by Brandt et al. [6], but with the benefit of increased confidence in classification accuracy afforded by the added context (the query, query refinements, the search engine, the annotation, and programming activity labels were all used to infer search motivations).

With reference to the second point, though much of the coded characteristics failed to yield eventual insights, an opportunistic approach was taken whereby the researcher coded for as many potential themes as possible [8].



Figure 3.8. Card sorting to discover search goal clusters.

*Card sorting.* When data collection was completed, all annotated searches were clustered to form distinct classes of search goals. To this end, card sorting was carried out using printed Trello cards (Figure 3.8). During this process, the search motivations initially coded were reviewed and further refined, acting as a kind of axial coding stage [36]. Though card sorting is typically employed to evaluate website content and navigation structures [11, 30], we found the iterative grouping and naming approach useful in cluster discovery.

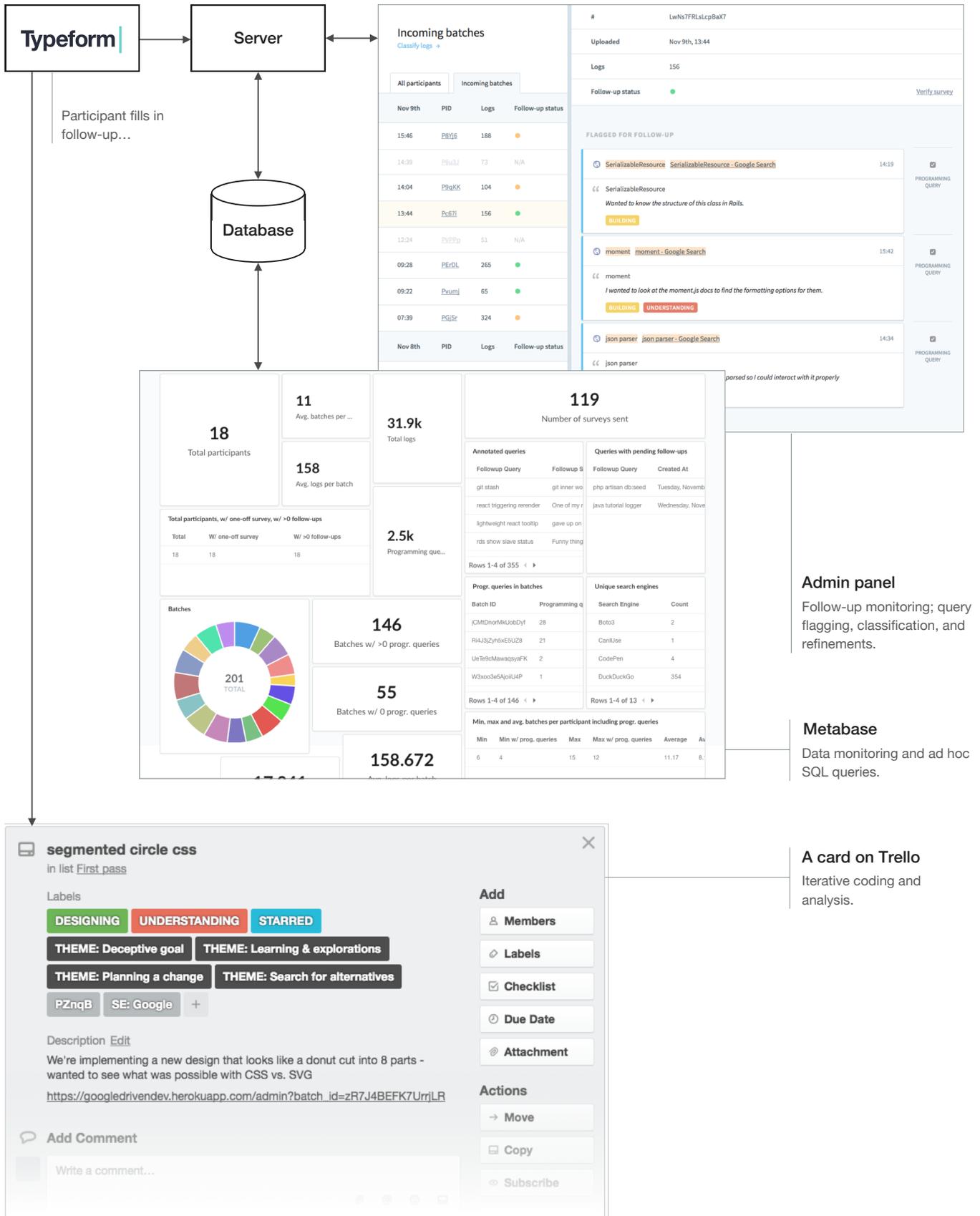
#### 3.4.2 Thematic analysis of exit interviews

Audio from the exit interviews was transcribed verbatim, and thematic analysis was conducted on the content. Snippets from

<sup>12</sup><http://metabase.com>

<sup>13</sup><http://trello.com>

<sup>14</sup>More formally, Braun and Clarke [8] describe the first class of themes as *semantic* and the second as *latent*.



**Figure 3.7. The data monitoring and preparation workflow.** Queries were flagged and logs classified on the admin panel. Data volume and follow-ups were monitored on a Metabase dashboard, which also allowed for ad hoc SQL querying. New follow-up responses on TypeForm triggered the creation of cards on Trello, ready for coding and analysis.

each interview were then organised into a code matrix for a high-level overview and for ease of comparison (Appendix I).

In the next two chapters, we present the results of these analyses separately: Chapter 4 discusses our findings on programmers' online search patterns, while Chapter 5 focuses on evaluating the IP-ES method.

#### 4. FINDINGS: DEVELOPERS' ONLINE SEARCH PATTERNS

This chapter presents our findings in two sections. To answer our first research question about programmers' Web search motivations, we identify seven search goal classes to complement those discussed in Chapter 2. To answer our second research question about the search strategies employed, we discuss aspects such as query construction and alternative search engines.

Before delving into the findings we provide a brief overview of the data corpus.

##### 4.1 The Data Corpus

The complete corpus consists of 31 893 logs collected from 18 individuals in 201 batches<sup>15</sup> (Appendix J). Of those logs, 2 488 were classified as programming searches: participants made an average of 15.9 programming searches per day (min = 1, max = 83, SD = 17.2). 357 programming queries were flagged for follow-up. Though 100% of the flagged queries were annotated by participants, seven were falsely flagged as related to programming, and participants could not recall the context for three. A total of 347 logs thus makes up the dataset of annotated queries.

##### 4.2 Search Goals

Drawing on the results of card sorting and thematic analysis of the annotated queries, the following classes of search goals were identified, ordered by prevalence (occurrences given in brackets):

###### **Hunting down just-in-time practical techniques.**

Practical *how-to*'s for immediate use where the specific APIs or language features are often unknown. (99)

**Understanding API or library particulars.** Further information about a known API or library, often concerning low-level mechanics. (82)

**Reminders.** Forgotten or obscure interface names or syntax. (62)

**Learning, research and investigation.** Best practices, trade-offs and popular opinion about a given technology, and introductions to new concepts. (30)

**Troubleshooting.** Fixes for an error or discrepancy. (29)

**Resources** Publicly available libraries to integrate into a project, programming or tool references, or other resources. (25)

**Navigation.** Shortcut to a known site. (17)

We were unsure of the category of two searches with insufficient context, and left one outlier uncategorised—a GitHub code search in the participant's own codebase checking whether an API had been used before.

Next, we discuss each goal briefly, providing examples from the annotated query dataset. For brevity, irrelevant extracts have been omitted from some annotations. Where it provides better illustration, some examples include query refinements from the same session. Unless otherwise indicated, all example searches have been made on Google.

##### 4.2.1 Hunting down just-in-time practical techniques

With almost a third of all searches falling under this category, finding practical techniques for immediate use was the most common search motivation among our participants. These *how-tos* concern the appropriate choice of unfamiliar APIs, combining several APIs, or wielding an API to a particular use case. For example:

[angular route uib tab] "Was wondering how to combine UIB library's tab directive with Angular UI-Router." (DSP7)

[react setstate sub property] "In React, state is immutable so you have to reset the state as a whole, but I wanted to mutate a sub key and set the state in one line." (DSP8)

The higher-level searches in this category often imply an unawareness of whether the solution would be a quick *one-liner* or an intricate mix of several APIs. The query therefore often describes the desired outcome in lay terms, as exemplified here:

[bootstrap button next to input] "I wanted to align a button next to an input field using Bootstrap." (DSP17)

##### 4.2.2 Understanding API or library particulars

A second common search goal concerns the particulars of an API or library. Though it occasionally overlaps with the search for practical techniques and reminders, the annotations given here often explicitly mention official documentation or codebases (indeed, programmers looking for a practical *how-to* may not necessarily care for the source, sampling resources opportunistically). In addition to generic documentation searches such as [php preg\_match] (DSP2), questions in this category include *Is this supported?* and *How has it been implemented?*, as exemplified below:

[forcelayout api] "Was searching for an ability to select parent node in D3 force-layout." (DSP9)

[golang copy built in] "I was trying to understand memory efficiency when [this function is] used with slices." (DSP11)

Absent in other cases, we also find searches here that occur when reading code authored by someone else:

<sup>15</sup>Recall that a "batch" refers to a participant's shared browsing logs from the past 24 hours.

[strlen] “I saw this php function in code and didn’t know what it did.” (DSP3)

[java comparator interface] → [java default string comparator]<sup>16</sup> “I was reviewing some code written by a consultant and was wondering why they were not using methods/classes from the standard library.” (DSP16)

Determining rationale in existing code has been found to be the most popular topic in code-related questions (e.g., *Why wasn’t it done this other way?*) [27], and we anticipated on-line resources to be unhelpful in these cases (instead, expert teammates are a typical source of information [27]). But our findings show that the Web is used, too, albeit on a small scale.

#### 4.2.3 Reminders

The third popular class of search goal was that of recalling forgotten interface names or syntactic details, often by explicitly seeking out examples. Searches here include:

[ubuntu search packages] “Searched after the command that’s used to search for programs on Linux Ubuntu.” (DSP3)

[URI uri = new URIBuilder] “To make sure the syntax is correct.” (DSP12)

DuckDuckGo: [java throw exception example] “Looking for the correct syntax for method error throwing.” (DSP1)

Broadly corresponding to the category described by Brandt et al. [6], the searches here act as quick lookups for obscure details or those deemed not worth remembering. Another similarity in the two studies, people often had a specific website in mind with these queries—they had likely made the search in the past and knew exactly where to look. In this example, “mdn” refers to Mozilla Developer Network:

[mdn transform origin] “CSS transform syntax.” (DSP2)

Targeting a specific resource may also explain the low occurrence of query refinement in reminder searches.

#### 4.2.4 Learning, research and investigation

30 searches were classed under the goal of general technology research (e.g., [job queue] (DSP6)), determining best practices (e.g., [bad things about scala implicits] (DSP14)) or investigating trade-offs, (e.g., [svg vs png] (DSP3)). Initially perceived as separate categories, there are subtle differences within this group between studying known concepts and learning about something new entirely. The former often occurs when planning a change:

[segmented circle css] “We’re implementing a new design that looks like a donut cut into 8 parts—wanted to see what was possible with CSS vs SVG.” (DSP7)

Learning about something new entirely, however, often occurs when embarking on an unfamiliar task:

DuckDuckGo: [best programming language for both ios and android] → [app dev design, what languages to use] → [code java app for ios and android] → [write helloworld ios app in java] “Been tasked with designing an app, no idea where to start.” (DSP1)

A third subtly different group is formed of searches that seem to exhibit casual curiosity, such as:

[boilerplate template] “My colleague was talking about boilerplate templates and that I should check them out. So I Googled to find out what they were.” (DSP3)

What links them all is a general lack of immediate actionability—information from various sources is gathered and internalised to be made use of in later lower-level tasks.

This high-level use case was rarely captured in the studies reviewed in Chapter 2. Though Brandt et al. [6] identify a learning category, too, we make a distinction here based on actionability. In our classification, their examples would likely fall under just-in-time practical techniques (e.g., [update web page without reloading php]). We suspect that the lack of a mindful planning stage in laboratory task set-ups (e.g., [6, 43]) prevents capturing cases of higher-level learning.

While participants were exploring unfamiliar concepts in these searches, most were within the general bounds of the expertise they had reported in the enrolment survey. And with regards to the few exceptions (e.g., the search by DSP1 above), searches were still opportunistic in the sense that the lack of understanding was a blocker to some immediate task. We conclude that cases of long-term learning—such as learning a programming language from scratch—are largely missing from our dataset.

#### 4.2.5 Troubleshooting

With similar prevalence to research and investigation, 29 searches were made with the goal of fixing errors or discrepancies. Unlike searches in other classes, the queries made here are consistently structured, often either describing the problem or copying the full text error message encountered, as in these searches:

[show is not a member of org.apache.spark.sql.GroupedData] “I didn’t know why I couldn’t show GroupedData. I wanted to find how to do it.” (DSP14)

[Possible infinite loop detected] “This was an error message from the percona tools I was using to sync up the replica instance with the master. I never dug into it too deeply though because I think it just required a better index value to work with.” (DSP4)

This makes them straightforward for a researcher to manually classify, especially when considered as part of a surrounding session.

It is also here that we encounter the most query refinement, with about half of the queries being refined at least once. For instance:

<sup>16</sup>We indicate query refinement here and in all future cases with [query] → [query].

[babel-jest] → [can't console log in babel jest] → [logging in babel-jest] → [jest-cli 16] “Thought I was in a different file than I actually was in so when running specs I couldn't get console.log to work.” (DSP5)

#### 4.2.6 Resources

Each of the above classes corresponds to the *informational* goal in the resource-navigation-information trichotomy of Web searches discussed in Chapter 2 [37]. But the remaining classes on our list—resource searches and navigational searches—correspond to the other two. Their low rate of occurrence reaffirms that programmers more frequently use the Web for informational searches compared to the general public (about 60% [37]).

In the majority of cases, participants sought third-party tools and libraries to integrate into the current project. For instance:

[json minify] “I wanted to find an online minifier tool to reduce whitespace in JSON. Standard JS minify wouldn't work.” (DSP10)

[rails kineses gem] “I'm looking at migrating our analytics pipeline to AWS and so I wanted to see if there were any gems to support this.” (DSP15)

The rest were searches for reference tables such as character codes or keyboard shortcuts (e.g., DuckDuckGo: [unicode] “Looking for a unicode table” (DSP1)), or resources such as dummy data. Searches in this category stand out for being just as likely to occur during coding as they are during higher-level planning and research activities.

#### 4.2.7 Navigation

Made largely out of convenience, navigational searches intend to reach a specific website (e.g., [mongodb] to navigate to mongodb.com (DSP18)). It is questionable whether they are really *searches*, but their prevalence illustrates the deceptiveness of queries isolated from context: at face value, many of them indicate information goals (misleading queries will be further discussed in Chapter 5). Of course, the end-goal of navigating may well be to learn about an API, as is likely the case here:

[fcm notification android example] “For just navigating to the firebase homepage.” (DSP12)

But classifying navigational queries based on their informational goal distorts the proportion of genuine searches where the desired website is unknown.

As a complementary summary to Figure 2.1 from Chapter 2, Figure 4.1 maps our search classes onto Ko, Myers and Aung's [25] learning barriers. Similarly to previous work, we found that most Web searches are conducted to overcome selection, coordination, and use barriers—these are the instances of picking up just-in-time techniques and studying APIs in more detail. However, we add a previously unnoted use of online search—the search for references and third-party

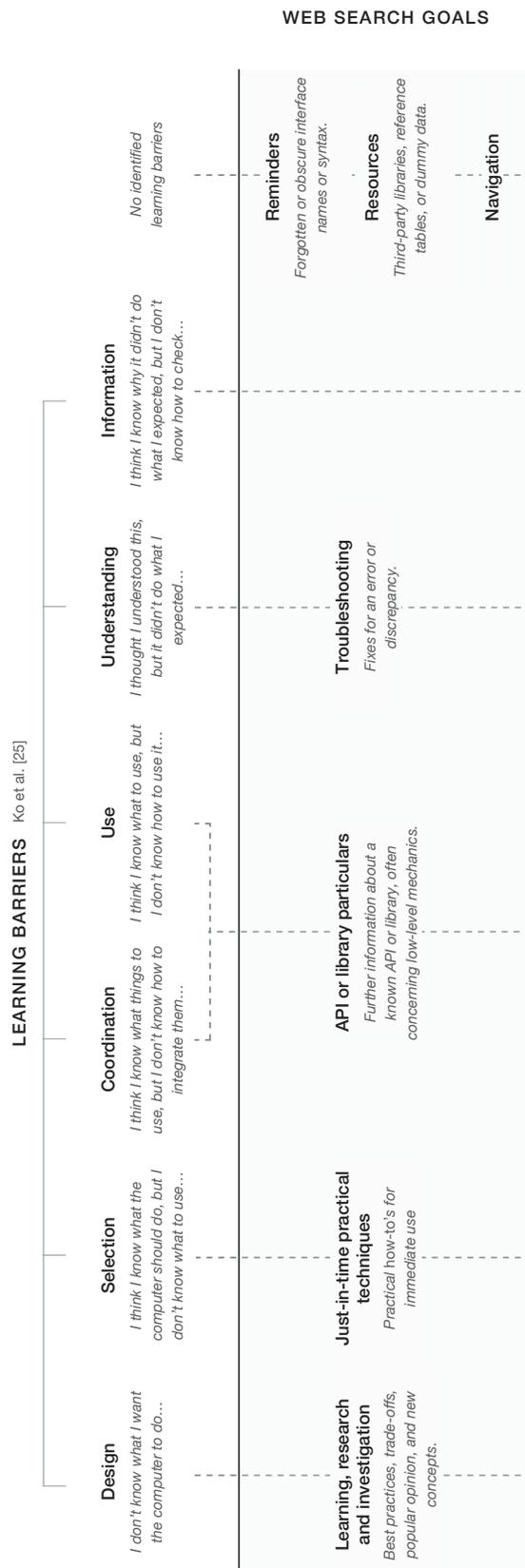


Figure 4.1. The seven goal classes of programmers' Web searches, mapped onto Ko, Myers and Aung's [25] learning barriers.

libraries—and include learning goals of a much higher level in our category of research and investigation.

The rest of this chapter discusses some specific search strategies used by our participants.

### 4.3 Search Strategies

#### 4.3.1 Choice of search engines

Though most people indicated the use of several alternative search engines in the enrolment survey, their proportional use in the log corpus was trivial (5%). Most participants used Google as their primary search engine (DuckDuckGo<sup>17</sup> was the search engine of choice for one participant), only seldom turning to specialised services to search documentation (e.g., Mozilla Developer Network<sup>18</sup>), code repositories (e.g., GitHub<sup>19</sup>), or language-specific symbols (e.g., Hoogle<sup>20</sup>). A single session may have included the use of multiple of these services, as seen here:

```
Google: [sumo collector filters] →
Google: [sumo collector filters include] →
Sumo Logic Community: [filters include] →
GitHub SumoLogic repository: [filters] (DSP4)
```

Few of the queries to documentation websites and symbol search engines were flagged for follow-up so we cannot comment on their context. But for the ten GitHub code searches that were annotated (out of 92 total), motivations varied from low-level behaviour investigations to source-code-as-documentation lookups.

Considering non-browser-based search tools, only one participant indicated in the enrolment survey that their IDE is equipped with search enhancements. Those who claimed to use offline documentation browsers (e.g., Dash) still made numerous Web searches for API particulars. Most participants instead described established workflows for rapidly switching between code and browser.

#### 4.3.2 Query construction

Comparing the use of natural language to the use of code terms in search queries, we encountered more code terms in reminder searches than we did in other classes, in line with previous studies [6]. 46% of queries in reminder searches contained code terms, compared to 38% in searches for API particulars and 15% in searches for practical techniques, whereas reminder searches contained the vast majority of queries with code terms *only*.

On the other hand, the use of grammatically correct natural language such as questions or complex noun phrase modifiers occurred, somewhat counterintuitively, similarly across all search goals:

```
[does md5sum read file into memory?]
(DSP15)
```

<sup>17</sup><http://duckduckgo.com>

<sup>18</sup><http://developer.mozilla.org>

<sup>19</sup><http://github.com>

<sup>20</sup><http://haskell.org/hoogle>

| Queries per session        |        |
|----------------------------|--------|
| 1 query per session        | 44.54% |
| 2 queries per session      | 43.7%  |
| 3 queries per session      | 13.73% |
| 4 queries per session      | 12.61% |
| > 4 queries per session    | 12.04% |
| max queries per session    | 10     |
| avg queries per session    | 1.87   |
| stddev queries per session | 1.56   |

Table 4.1. Queries per session.

```
[what determines if a site is considered
intranet for ie compatibility view
settings] (DSP16)
```

```
[javascript library that embed a string
and if find an embed content render] (DSP2)
```

#### 4.3.3 Query refinement

Table 4.1 shows that most search sessions were made up of a single query, while the average number of queries per session was 1.87. This indicates more query refinement than previous reports of 1.45 queries per session among programmers [6], but less than the 2.02 made on average by the general public [40]. We are hesitant to draw any conclusions: the annotated query dataset contains too few searches for any meaningful statistical analysis of query refinement compared to the large volumes of aggregate log analysis.

The context included, however, allows peering further into the qualitative aspects of query refinement among our participants. In addition to differences across goals discussed in Section 4.2, we find sessions that start with general and descriptive queries and end with specific ones as the programmer discovers appropriate APIs (a practice also reported in past studies [20]). For example:

```
[postgres export database schema] →
[psequel view schema] →
[pg_restore] → [pg_dump] (DSP17)
```

We would also like to note one particular case of query refinement where search language itself is altered:

```
[step by step json lexer] →
[comme construire un lettoire json] →
[como crear lexer json] (DSP1)
```

Two participants indicated in the enrolment survey that they occasionally make use of languages other than English in programming queries. One notes:

```
“Results in Spanish tend to have more extensive explanations. Results in Italian or French yield fewer results typically.” (DSP1)
```

With English being the mainstream language of the IT industry, as well as the standard in most programming language

keywords<sup>21</sup>, it is rational for non-native speakers to seek out English resources. However, the occasional use of other languages has implications for studies on programmers' Web use. Capturing these cases and comparing them to English-based browsing can reveal how programmers evaluate resources—especially when found in query refinements.

#### 4.3.4 Cross-domain translations

Occurring primarily in reminder searches, using analogies to recall API methods was common among participants, as noted also in previous studies [6, 17, 43]. In these cases, the utility of Web search in addressing the “vocabulary problem” [43] is evident, and enables unplanned learning.

The approach is likely mostly subconscious: people simply query using familiar terminology. We therefore typically relied on annotations and query refinements when tagging searches with this label. For example:

```
[jquery add to list] →  
[jquery add to array] (DSP3)
```

However, deliberate use of analogous terms occurred, too, where participants mentioned the corresponding APIs explicitly:

```
[mongoid update_attributes] “Was looking for the  
equivalent of a Rails ActiveRecord method in the Mon-  
goid persistence ORM.” (DSP18)
```

#### 4.3.5 Reliance on social validation

A prominent theme drawn from annotations which has not been discussed in past studies on programmers' search practices is the importance of social proof and best practices. Intuitively, programming is, on the lower-level, an objective field where the choice of APIs or methods is either appropriate or inappropriate (less so, of course, when it comes to higher-level design choices). Still, mentions of “the best” or “the accepted” way of doing things were frequent in questions of practical techniques, API use and technology research alike:

```
[react pass state to child] “I wanted to pass the  
entire react state object to a child component, this is fairly  
simple to ‘just do’ but I was looking for the accepted and  
‘safe’ way of doing so.” (DSP8)
```

```
[best process supervisor] “I was ... looking for  
people’s opinion on Linux process supervisors/monitors  
for managing web processes, workers, etc.” (DSP6)
```

```
[mysql using index] →  
[mysql indexes best practices] (DSP2)
```

## 5. METHOD EVALUATION

Having discussed our findings on programmers' online search patterns, we now turn to a methodological review. The study criteria established dictated that the study should (1) capture real world context, (2) be unobtrusive, (3) allow for anonymity from the investigator, and (4) include participant commentary. To evaluate the IP-ES method against these criteria, we present themes drawn from exit interviews alongside supporting meta-data from the log corpus. We also illustrate the investigator's

perspective, discussing practical issues of conduct and analysis and potential future applications. But to begin, we present a brief overview of participation rates and data volume.

### 5.1 Participation Rates and Data Volume

22 people responded to an invitation to participate that was sent to a total of a 107. Two of those initial participants withdrew from the study on the first day having shared no batches, while the rest took part for the full two weeks and responded to all follow-up surveys. The data for a further two participants were deleted on account of consistently delayed survey responses and difficult-to-interpret annotations, resulting in an eventual total of 18 participants. This is well below the potential scale of a traditional instrumented panel, and is regarded as insufficient for improving search engine heuristics [17]. However, being interested in the qualitative aspects of each search, we consider the participation rate adequate.

While a maximum yield of 756 search queries could have been flagged with this number of programmers (three queries a day for two weeks), less than half of this (357) was flagged and annotated in reality. Firstly, 27 batches contained less than three programming queries from separate search sessions while 55 contained none; participants therefore received a follow-up survey only 60% of the time on average (see Appendix J). Secondly, the expectation of three annotations per 24 hours could not be strictly enforced. The standard 24-hour sharing interval was prolonged during time spent away from the computer—for example, a batch of three days' logs would be shared on a Monday morning after a weekend offline. This occasionally resulted in follow-up requests per multiple-day-long periods: 6% of batches had an interval length of two days or more ( $M = 25.9$  hours,  $SD = 15.23$  hours). It should be noted that though these mechanisms prevented us from realising full potential in annotation volume, they also helped the system to self-regulate, allowing it to adjust to each participant's routine.

### 5.2 Participant Perspective

#### 5.2.1 General study set-up and participant burden

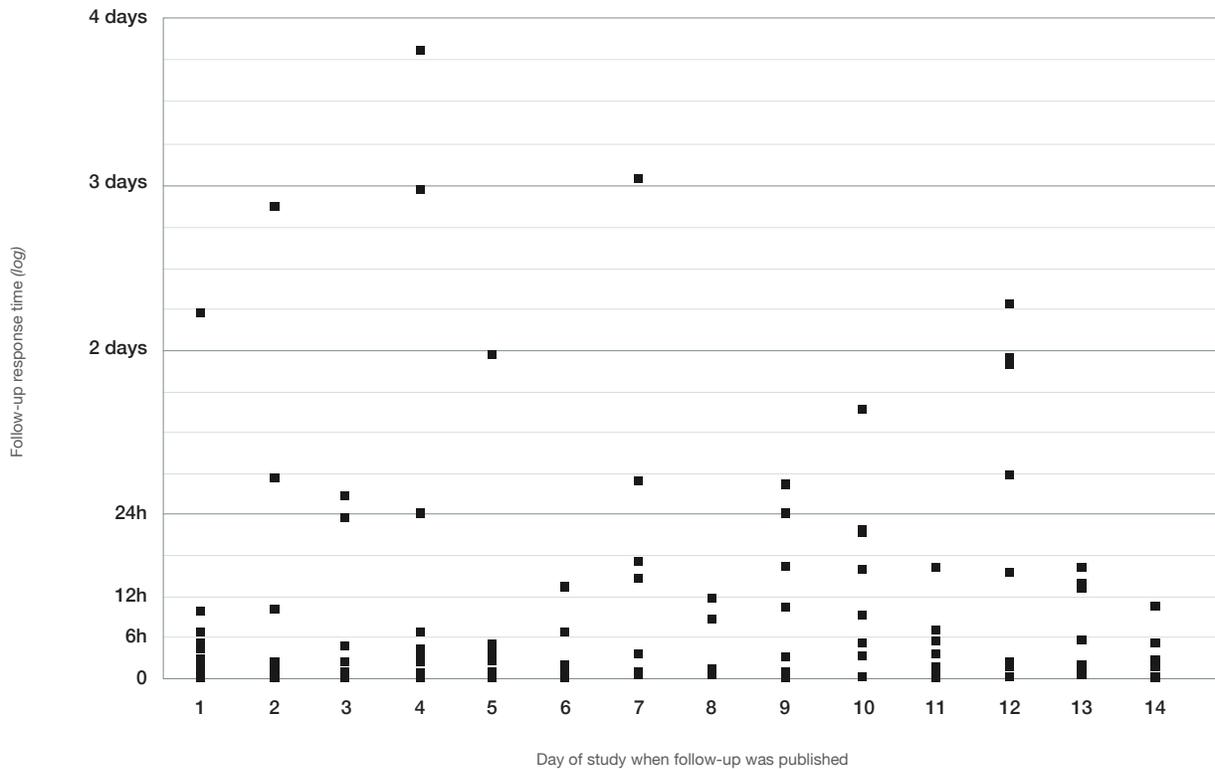
We begin the discussion of interview themes by considering the general study set-up and participant burden. Most interviewees found the data collection well organised and undemanding, demonstrated also by the high follow-up response rate. The average time taken to complete a single follow-up was 6.8 minutes, while the median was 2.5. This largely matches participants' perception of commitment required. For example, one interviewee remarks:

```
“It only takes like two minutes. Or maybe three minutes.  
... It’s not a huge commitment. I think the evening time  
was perfect because productivity is kind of plummeting  
... so spending two minutes answering a survey isn’t a  
huge deal.” (IP1)22
```

Participants were also appreciative that no special set-up was required for individual search routines. Past empirical software engineering studies have largely focused on Eclipse users

<sup>22</sup>“IP” refers to Interview Participant. Though all interviewees took part in the instrumented panel, we use separate notation here, as participant anonymity prevented us from knowing which instrumented panel user a given interviewee represented.

<sup>21</sup>Origin country search on <http://hop1.info/>.



**Figure 5.1.** Follow-up survey response times for each of the 14 days in the data collection period (min = 5 minutes 14 seconds, max = 10 days 17 hours, SD = 28.23 hours).

(e.g., [16, 24, 33]) (presumably because of the ease in which loggers and plugins can be written for the IDE), and this limits the variety of behaviours expressed in the sample. The use of a browser extension can accommodate a range of search behaviours, as this comment illustrates:

“Because I use DuckDuckGo as my primary search engine it recorded it anyway, it was very easy for me to follow up.” (IP6)

We believe our aim of capturing real-world context was largely achieved. Interviewees claimed that they generally forgot they were being tracked, often citing a state of flow when programming. For example:

“I think I generally forgot. . . . Especially with programming, you get so in- like, the zone with what you’re coding that...” (IP7)

This unawareness was usually broken only when they were prompted for follow-ups, as one interviewee notes here:

“As soon as I installed it, I forgot about it . . . . And then, I remembered again once I saw the little icon light up with another follow-up survey. But after I’d done that, I forgot about it again.” (IP1)

It is therefore unlikely that participants made any significant changes to their usual search practices during data collection; the few exceptions are discussed in Section 5.2.5.

### 5.2.2 Follow-up reminders and response delay

Timely follow-up completion was another important point of consideration. Section 3.3.2 describes the efforts made to publish follow-ups promptly to avoid biased retrospection. The extra delay from publishing to completion was mostly insignificant: follow-ups were completed within a median of 2.61 hours of receipt, with 32% of responses falling below a 1-hour delay and 64% falling below a 6-hour delay (Figure 5.1).

12 participants also opted to receive additional survey reminders by text message, but most interviewees considered the plugin notification sufficient, often citing familiar routines in the use of the widget area:

“I think the notification in the icon alerted me anyway because I’ve got a couple of things installed there that alert me so I’m aware of what’s going on in that top right corner.” (IP3)

“I have a couple of other extensions there and ( ) when it’s time to do feedback I fill it in... because it’s illuminated.” (IP5)

In contrast, the text messages were not considered useful due to inconsiderate timing:

“You send me a reminder, and I think about it and I don’t have the questions for another 12 hours or so.” (IP2)

“I don’t think I really needed [the texts]... but that’s just cause I was working [at my computer] the whole time.” (IP7)

“I never actually looked at the text and then went to do the follow-up. I always saw the icon change and then did the follow-up.” (IP6)

Daily routines were also cited as helpful in managing the surveys, but because of the varying batch intervals discussed above, this strategy is only effective where schedules are stable, triggering follow-ups at similar times each day:

“I guess the first thing in the morning I just noticed the little notification and it just became a part of my things to do . . . before I actually start work, like checking emails, checking Slack, things like that.” (IP3)

### 5.2.3 Recall accuracy

We anticipated a fall in recollection ability with longer follow-up delays: in the pilot, the participant found it difficult to remember his motivations for a search made the day before. But in the main study, participants forgot the context of their search in only three cases (these searches had occurred 21 hours, 3 days and 4 days prior to being annotated). Most interviewees claimed memory had rarely been an issue, and some cited query structure as a helpful hint:

“Generally, the [query] was just what I’d been searching for so I could just say ‘Looking for...’ whatever the search was. Mostly, . . . it was self-explanatory.” (IP5)

“There’s this sort of format that you write . . . which probably helps give a lot of context because you’re normally like ‘the language’, ( ), ‘the problem’ . . . so you kind of get this sentence structure in your searches. So I guess that’s . . . what makes it easy enough to remember.” (IP7)

While most annotations were of high quality and described context beyond what was visible from the query alone, this reliance on the query itself to describe search goals was sometimes evident. For example:

[jackson json validation] “how to validate json using jackson” (DSP14)

This suggests that participants either failed to recall exact context and resorted to superficial reasoning more often than consciously acknowledged, or, indeed, that they opted for brevity due to lack of time or interest:

“Some days I had stuff to do at work and then I had to fill them in . . . fast. Other days when I was bored at work and I didn’t have that much . . . to do I tried to write more proper sentences.” (IP9)

To avoid pitfalls related to recall accuracy, future iterations of the browser extension could only display survey links for a predefined period [4].

### 5.2.4 Understanding follow-up questions

When discussing follow-up questions, interviewees voiced no issues with comprehension. Deciding on the appropriate

| Activity label    | Used on % of queries |
|-------------------|----------------------|
| Building          | 44.54%               |
| Understanding     | 43.7%                |
| Managing overhead | 13.73%               |
| Designing         | 12.61%               |
| Testing           | 12.04%               |
| Editing           | 10.92%               |
| Other             | 5.6%                 |

**Table 5.1.** The proportional use of programming activity labels to tag search queries. Note that the total exceeds 100% because a query could be tagged with multiple labels.

programming activity labels in Question 2 (see Table 3.2), however, was challenging to many:

“The difficulty was in putting them into categories. Because I might be searching for something sort of... tangential to what I was doing. . . . So it was very difficult to work out what to actually put there. . . . I might have been testing stuff, but [discovered suddenly that] ‘Oh, actually I need to fix this’. You start fixing it and you start Googling, and... Well, I *was* testing, [but] I’m still *now*, you know.” (IP1)

“So, for me it fitted into two sort of areas, because all the work I’ve been doing for the past couple of weeks has been similar. . . . When it said ‘Testing’ I.. I mean I’m writing tests all the time. My workflow is testing. So it wasn’t really relevant.” (IP4)

The difficulty was twofold: participants cited conflicts between the activity label and their mental model of it, as well as the activity label and what they had actually been doing (e.g., when multitasking). As seen in Table 5.1, tagging queries with “Building” and “Understanding” was particularly popular. Using the same class of activities in a survey, LaToza and Myers [28] reported a more uniform distribution. This may, of course, signify that Web search simply occurs more often during “Building” and “Understanding”. But given the confusion expressed among interviewees and examples of obscure use in annotations (such as “Designing” taken to mean visual design, rather than its intended sense of architectural planning), one must also consider the possibility that they were interpreted as high-level activities that apply to most cases.

For this reason, activity labels alone were rarely taken as a basis for analysis. They were, however, useful in supplying additional context when coupled with annotations and the surrounding browsing sessions. A review of how the activities are termed would benefit future studies.

Finally, while not directly related to question comprehension, some minor issues were raised with TypeForm’s survey interface (e.g., hover effects, keyboard navigation). The use of a third party inevitably reduces researcher control, but we maintain that the ease of integration with an established service outweighs the ability to fine-tune interface elements.

### 5.2.5 Privacy and response to being tracked

Privacy concerns were anticipated to be the biggest barrier to participation. In this regard, the group of programmers participating may have largely been self-selecting: the two participants who withdrew cited unease with tracking, while those interviewed were generally indifferent. Elaborating on their approach, they often took the stance that privacy in today's online world is largely an "illusion" (IP2):

"I'm very much of the opinion that everything's tracked by something somewhere anyway so ... tracking-wise it doesn't particularly bother me." (IP3)

"I'm not the most concerned person about privacy to be honest. ... I can't kid myself with privacy when I know I put so much of my information online anyway." (IP6)

The use of multiple Google Chrome profiles or separate work computers may also have advocated a relaxed approach to privacy. In these cases, browsing unrelated to work is hidden from the plugin. Seven interviewees already had these multi-profile practices in place prior to the study.

While we anticipated a high demand for the log filtering options provided (see Section 3.2), little use was reported in exit interviews: only four individuals specified a logging time-frame or added URLs to the blacklist. Moreover, they claimed to do so not because of privacy concerns but simply to reduce the volume of irrelevant logs. (An example of a similar courtesy to the investigator, one participant claimed to make conscious efforts to search the Web more frequently than usual, though only for the first few days.) A single interviewee made use of the batch download option. Still, the availability of these features was appreciated:

"Well, the blacklisting feature is very very good and I used it. But I used it mostly to keep the number of logs down, for example, sites like Reddit and Amazon and stuff, there's no value in them being in the logs." (IP1)

"The only tracking during certain hours and the blacklisting were nice I think. ... I looked over the defaults you put in and they all seemed kind of okay. So I didn't add anything else to it. But yeah, that was nice to see ... " (IP7)

"It wasn't about of privacy, it was more that you wouldn't have to go through so much data." (IP9)

On the other hand, this indifference to privacy was not reflected fully in participants' response to being tracked, as some cited increased awareness of their search topics, ranging from casual musings to feelings of shame:

"I generally tend to search for syntax, so being a developer for 9–10 years now, it kind of makes me feel dumb that I'm actually searching for really basic stuff." (IP2)

"Sometimes—because it's my first year programming—I would ask what I think are quite stupid questions ((laughs)). So I was wondering 'oh dear, I hope I don't have to follow up on this question' ((laughs))." (IP6)

Some past research has indeed made the assumption that expert programmers make fewer reminder searches (e.g., [6]). In our dataset, however, programmers of all experience levels were likely to search for syntax reminders.

It should also be noted that negative feelings were conjured less by the tracking itself and more by having to follow up on certain searches:

"The questionnaire is what gets me the most because whenever I answer the question, I'm like 'I can't believe I was that dumb to actually not know the answer to that'." (IP2)

"The fact of searching for it is fine, it's when you have to *explain*... ((laughs)) why you were searching it." (IP6)

Though these musings remained mostly passive, in the case of one participant, some syntax searches were made in Incognito mode specifically to conceal them from the logger. As the participant later elaborated, this only occurred during the start of the study:

"The hassle of actually [using] Incognito to do my shameful searches for syntax kind of disappeared after the first day or so." (IP2)

Most interviewees, however, did not feel self-conscious about specific search topics and many cited the experience of reflecting on one's work as enjoyable:

"I actually quite liked it because getting the follow-ups every day meant that I have to reflect back on the day and think about what was I doing, which was quite nice." (IP1)

"I found it funny looking back at what I'd searched for the previous day, just quite entertaining." (IP3)

"Every single one was kind of interesting because it was like, 'oh, this day I worked on stuff that was Ruby, this day I worked on stuff that was PHP, this day it's MySQL, this day it's Mongo'. It was kind of interesting to see how the focus of my work changes depending on the day. It wasn't something I'd realised before." (IP8)

An emotional engagement with the follow-up surveys was also sometimes evident in the annotations: some reiterate their discoveries at length (an opportunistic learning strategy?) while others express frustration:

[hook-cron] "I have somewhat of an idea of how cron.php works in a Drupal server, but the documentation online for Drupal 6 is non-existent. In fact, I get an un-styled html page that says 'File not found' when I click on the docs... useless." (DSP4)

A full account of exit interview codes and themes is given in Appendix I.

### 5.3 Investigator Perspective

To evaluate our method from the perspective of the investigator, we now turn to factors of a more operational nature:

investigator burden and the utility of annotations. We also discuss some alternative IP-ES applications, showing that the approach is not constrained to studying search behaviour.

### 5.3.1 Investigator burden and study scale

During data collection, the investigator’s main function was to flag queries for follow-up. Facilitated by the random sampling system built into the admin interface, surveys were mostly published within minutes. More demanding, however, was managing the high frequency of incoming batches. With an average of 10 batches shared per day across different time-zones, notification emails and mobile availability of the admin interface were instrumental.

Though only three queries were flagged at a time, all logs were eventually classified as “programming query” or “other browsing”. This was less time-sensitive as overall proportions played only a minor role in our analyses (they were more helpful in assessing methodological potential). Still, the tailor-made system allowed us to design an efficient workflow: whereas during the pilot logs had been classified one by one from a single pool, we later implemented per-batch classifications where just the programming queries were explicitly classified and everything else was “other” by default.

Similarly, a custom-built interface eased the burden of verifying query refinements. Here, the admin panel displayed all queries in a session one by one (automatically extracted based on the 5-minute interval rule; see Section 3.4), and the investigator indicated the correct number of refinements with a keystroke. As most sessions contained a single query, the process took less than an hour to complete.

In larger-scale studies, query flagging and log classification could be handled with further automation or more human resource. Card sorting, however, would be problematic: with several hundred searches, it was both operationally and cognitively demanding. In these cases, conducting multiple card sorts with subsets of the data has been shown to be feasible [30].

### 5.3.2 The utility of annotations

As illustrated by many of the example searches given in Chapter 4, the main conclusion we draw from analysing annotations is that queries alone can be deceptive when inferring search goals. They may indeed reliably predict intent in non-specialist searches as previously suggested [37]. But in the case of programming, similar query structure is used with a range of search motivations; that is to say, similar query structure is not reserved for obviously similar search classes like *how-to* and documentation searches. Consider, for example, the following generic query: [javascript infinity]. It could suggest a low-level syntax search for the infinity symbol in JavaScript, but also a high-level learning goal, as was the case here:

[javascript infinity] “Learning what infinity is used for.” (DSP9)

Similarly, queries that contain high-level concepts and thus indicate a learning goal could actually seek help with syntax:

[react component lifecycle] “I wanted to find the docs relating to the react component lifecycle as I forgot componentDidMount().” (DSP8)

Furthermore, 19 annotations were coded with an explicit search for an example, whereas only five of them included the word “example” in the query.

Numerous annotations referenced coworkers and the social context of programming, creating another surprising category of deceptive searches. Here, the annotation illustrates cooperation:

GitHub: [remote disconnected] “Someone on my team thought they found a bug in probably the most popular nodejs mongo drivers. I didn’t find anything that indicated such. . . .” (DSP4)

Another example of social context, resources were sometimes sought not for learning, but teaching:

[easing functions examples] “To showcase to my fellow developer what easing functions are and to show visual examples.” (DSP10)

We therefore believe the use of annotations leads to more accurate search goals compared to studies where no first-hand commentary is given. For example, misleading queries may partly account for why the high-level use case of research and investigation has rarely been captured in log analyses in the past (e.g. [16, 20]). The analysis of Stack Overflow questions [44] was the only study reviewed that reported information goals at a similarly high level—and in this case, plenty of context was available (forum posts must include thorough background to attract good answers).

On the other hand, even with annotations, search goals were not always clear. The follow-up question (*What triggered this search?*) was deliberately open-ended so as to prompt serendipitous comments, but this may also have attracted some low-quality annotations. Others provided detail that was irrelevant to the motivation of the search, as seen here:

[mongouri config] “What I was really searching for here was ‘connection string’, but it wasn’t coming to my head, so I floundered for a good while before I figured out what I really needed.” (DSP4)

In these cases, interpreting search goals relied somewhat on personal judgement, but more heavily on supporting data such as the programming activity labels the query was tagged with, or browsing logs in the surrounding session. It therefore seems unlikely that the rare ambiguous annotations would significantly affect our analysis.

A factor that could have a significant impact on analysis, however, is investigator judgement when deciding which queries to flag in the first place: if logs are falsely classified as programming searches, the annotation is rendered unusable. The classification issue has been previously addressed by using only an obvious subset of the query log, e.g., searches containing the word “java” [20]; as shown, this would exclude a sizeable number of non-obvious programming queries. In this study, we took measures against false classifications by taking

note of the technologies used by each participant (reported in the enrolment survey). Still, at least seven queries were falsely classified as related to programming. The number of false negatives is potentially higher as non-programming-related logs were never annotated and thus never audited.

### 5.3.3 Alternative applications

The current study focused on what programmers search for online, but annotations often also referred to programmers' work practices and the cultural and social contexts they operate in, possibly making the method a useful addition in ethnographic studies, if appropriately adapted. The frequent references to coworkers discussed in the previous section provide one example; as another, here, a participant describes their strategy for programming in an unfamiliar language:

"I'm programming something in golang, but I don't have enough familiarity with the language to achieve my goal with any amount of speed, so I'm implementing an algorithm in python before I attempt to port it over to the golang code." (DSP4)

And here, we get a cultural glimpse into a development team's communications:

[git --force] "Making a nerdy star wars + git joke on slack." <sup>23</sup> (DSP7)

Abundant data can also be drawn from the full log corpus, confirming the utility of browsing logs that retain user and session information [17, 46]. Yielding among other things session lengths, site visits during a search session and search parameters<sup>24</sup>, a variety of usage patterns can be examined beyond what the current paper has discussed.

### 5.3.4 Participant communications

As a final point of discussion in our methodological review, we stop briefly on an unexpected managerial issue: the lack of personalised communications with participants. The investigator had, *de jure*, no ways of associating a log trail with a known individual, so personal feedback or checking in with participants who fell behind on surveys was challenging. We continue to value participant anonymity from the investigator, but the loss of these ad hoc communications may impact data quality. The browser extension could certainly be equipped with, say, an anonymous, per-participant messaging system. Given the indifference to privacy expressed by interviewees, it is uncertain whether this effort would be worthwhile.

## 6. CONCLUSION

In this report, we have presented our findings on what programmers in the wild search for on the Web, and what strategies they employ during the process. We have also presented a novel data collection method, dubbed IP-ES, that draws on techniques from query log analysis, diary studies and experience sampling, demonstrating its use with a Chrome extension.

<sup>23</sup> [git --force] in this context also provides an example of a falsely classified programming query.

<sup>24</sup> Query parameters in Google URLs, for instance, distinguish typed searches from Google Instant searches.

We conclude with a discussion of three insights into programmers' Web search practices that have wider implications for design, and how the IP-ES method can be helpful in uncovering similar findings in the future.

**Web search is leveraged during a range of programming activities.** Past studies have focused on Web use during activities that involve actual coding, resulting in categorisations that revolve primarily around documentation and reminder searches. While developers arguably spend the majority of their time on lower-level programming activities, our findings illustrate that Web search is equally leveraged during higher-level activities such as planning a feature or investigating potential technologies.

Participants' commentary revealed that reading code authored by other people also occurs as a prevalent work activity, and one where Web search plays a role. Often representing a barrier to information as per Ko, Myers and Aung's terminology [25], confusions that arise during code reviews or when dealing with legacy code are typically resolved by simply *examining* code or consulting team members. Our findings demonstrate that online documentation lookups are similarly helpful. In these cases, programmers will construct their query from found code snippets or specific API names.

**Reminder searches occur frequently despite ample opportunity to avoid them.** Syntax lookups are arguably among the easiest to automate as the information sought is atomic. The need for basic reminders is already addressed by several tools described in the literature [5, 20, 32, 43], and by simple IDE features such as auto-completion. Yet, few participants reported using such tooling, and nearly a fifth of the searches in our dataset fell under syntax reminders. We believe there is ample opportunity to provide syntax reminders with IDE enhancements, prompting the question: Are there no adequate tools available? Or has accessing Web resources become so convenient that any alternatives would fail to provide considerable benefits? If so, perhaps future design efforts should be focused not on isolated IDE enhancements, but on improving the algorithms and heuristics of general Web search engines.

**Social validation is valued in questions of coding.** The need for social validation may be more important in questions of coding than previously thought. Intuitively, the search for opinion is more common during high-level planning activities, but our findings show that social proof is also sought when facing lower-level coding issues, such as API use. This has implications for the design of search tools that filter and re-format search results such that only interface names and code snippets remain (e.g., [20, 43]); while effective for syntax lookups, presenting information in this way makes it harder to gauge best practices.

We would also like to present three key learning points from a methodological perspective.

**Query logs alone are insufficient for analysing user goals.** Instrumented panels can be seen as a compromise between the rich, contextual data of an observational study and the high-

volume yield of an aggregate log analysis [17]. By asking programmers to annotate their logs, however, we have shown that contextual information increases not only the richness of data, but—crucially—validity, too. Programming queries lacking commentary can be misleading, and though broad-brush heuristics (such as the use of programming terms in a query) can be applied to query logs consisting of millions of items, researchers should not rely on them for smaller datasets where critical nuances can be missed.

**Programmers are comfortable engaging with a browser plugin.** With instrumented panel participants facing long-term demands to provide timely information, it is beneficial for the medium to integrate conveniently into the user’s work environment. We have demonstrated a browser extension to be suitable for these purposes, as programmers are familiar with the paradigm and thus feel comfortable engaging with the study. Though some participants were self-conscious about their searches and privacy concerns should certainly not be understated, we believe using a familiar medium can at least alleviate them.

A browser plugin is also intimately tied to Web use. A less discreet piece of software that has to be separately installed may disturb existing workflows and carries the risk of neglect. It would also fail to accommodate any existing strategies people use for separating work and personal browsing, such as the use of multiple browser profiles.

Finally, although many participants are excluded if the plugin is made available on a single browser, Google Chrome’s popularity makes it a low-risk choice in 2017: 80% of respondents to our recruitment survey were Chrome users.

**Anonymity from the researchers is valued by participants, but may not improve participation rates.** Assuming that programmers can be persuaded to take part in the study by alleviating concerns over privacy and anonymity led us to expend significant technical efforts on added features. However, we found from interviewing participants that attitudes towards privacy are largely established (even if they are relaxed), and a self-selection bias apparent among our participants suggests that any efforts to sway these attitudes with added anonymity may have trivial gains.

As the software industry faces increasingly sophisticated requirements and opportunistic practices continue seeping into programmers’ workflows, it is clear that the Web will carry on as an indispensable resource for software engineers. Our premise has been that helping them navigate this network of information, formulate the right questions and home in on the relevant presents a high-impact opportunity of improving software quality as well as work satisfaction. This is an ambitious project, but one that relies fundamentally on an understanding of how programmers today leverage the Web. In this report, we have highlighted search behaviours that can help mould tool design around existing mental models, and presented one potential approach to the continued study of programmers’

Web use in the wild. We await excitedly for future findings and industry-tested design directions.

## ACKNOWLEDGEMENTS

I would like to thank my project advisor Nicolai Marquardt for his guidance and support in carrying out this research; Karl Sutt for his technical advice in building the data collection system; and all of the participants for expressing their openness and sharing their insights.

## REFERENCES

1. Eytan Adar. 2007. User 4xxxxx9: Anonymizing query logs. In *Proc of Query Log Analysis Workshop, International Conference on World Wide Web*.
2. David E Avison and Guy Fitzgerald. 2003. Where now for development methodologies? *Commun. ACM* 46, 1 (2003), 78–82.
3. Sushil Bajracharya, Trung Ngo, Erik Linstead, Yimeng Dou, Paul Rigor, Pierre Baldi, and Cristina Lopes. 2006. Sourcerer: a search engine for open source code supporting structure-based search. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*. ACM, 681–682.
4. Niall Bolger, Angelina Davis, and Eshkol Rafaeli. 2003. Diary methods: Capturing life as it is lived. *Annual review of psychology* 54, 1 (2003), 579–616.
5. Joel Brandt, Mira Dontcheva, Marcos Weskamp, and Scott R Klemmer. 2010. Example-centric programming: integrating web search into the development environment. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 513–522.
6. 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*. ACM, 1589–1598.
7. Joel Brandt, Philip J Guo, Joel Lewenstein, and Scott R Klemmer. 2008. Opportunistic programming: How rapid ideation and prototyping occur in practice. In *Proceedings of the 4th international workshop on End-user software engineering*. ACM, 1–5.
8. Virginia Braun and Victoria Clarke. 2006. Using thematic analysis in psychology. *Qualitative research in psychology* 3, 2 (2006), 77–101.
9. Andrei Broder. 2002. A taxonomy of web search. In *ACM Sigir forum*, Vol. 36. ACM, 3–10.
10. Margaret Burnett. 2009. What is end-user software engineering and why does it matter?. In *International Symposium on End User Development*. Springer, 15–28.
11. Alan Cooper, Robert Reimann, David Cronin, and Christopher Noessel. 2014. *About face: the essentials of interaction design*. John Wiley & Sons.

12. Mihaly Csikszentmihalyi and Reed Larson. 2014. Validity and reliability of the experience-sampling method. In *Flow and the foundations of positive psychology*. Springer, 35–54.
13. Steve Easterbrook, Janice Singer, Margaret-Anne Storey, and Daniela Damian. 2008. Selecting empirical methods for software engineering research. In *Guide to advanced empirical software engineering*. Springer, 285–311.
14. Norman Fenton, Shari Lawrence Pfleeger, and Robert L Glass. 1994. Science and substance: A challenge to software engineers. *IEEE software* 11, 4 (1994), 86.
15. Thomas Fritz and Gail C Murphy. 2010. Using information fragments to answer the questions developers ask. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*. ACM, 175–184.
16. Max Goldman and Robert C Miller. 2009. Codetrail: Connecting source code and web resources. *Journal of Visual Languages & Computing* 20, 4 (2009), 223–235.
17. Carrie Grimes, Diane Tang, and Daniel M Russell. 2007. Query logs alone are not enough. In *WWW 2007: Workshop on Query Log Analysis*. Citeseer.
18. Björn Hartmann, Mark Dhillon, and Matthew K Chan. 2011. HyperSource: bridging the gap between source and code-related web sites. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2207–2210.
19. Björn Hartmann, Scott Doorley, and Scott R Klemmer. 2008. Hacking, mashing, gluing: Understanding opportunistic design. *IEEE Pervasive Computing* 7, 3 (2008), 46–54.
20. Raphael Hoffmann, James Fogarty, and Daniel S Weld. 2007. 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*. ACM, 13–22.
21. Bernard J Jansen. 2006. Search log analysis: What it is, what's been done, how to do it. *Library & information science research* 28, 3 (2006), 407–432.
22. Bernard J Jansen and Amanda Spink. 2006. How are we searching the World Wide Web? A comparison of nine search engine transaction logs. *Information Processing & Management* 42, 1 (2006), 248–263.
23. Irvin R Katz, Marian Petre, and Laura Leventhal. 2001. Editorial: empirical studies of programmers. *International Journal of Human-Computer Studies* 54, 2 (2001), 185–188.
24. Miryung Kim, Lawrence Bergman, Tessa Lau, and David Notkin. 2004. An ethnographic study of copy and paste programming practices in OOPL. In *Empirical Software Engineering, 2004. ISESE'04. Proceedings. 2004 International Symposium on*. IEEE, 83–92.
25. Andrew J Ko, Brad A Myers, and Htet Htet Aung. 2004. Six learning barriers in end-user programming systems. In *Visual Languages and Human Centric Computing, 2004 IEEE Symposium on*. IEEE, 199–206.
26. Craig Larman and Victor R Basili. 2003. Iterative and incremental development: A brief history. *IEEE Computer Society* (2003), 47–56.
27. Thomas D LaToza and Brad A Myers. 2010. Hard-to-answer questions about code. In *Evaluation and Usability of Programming Languages and Tools*. ACM, 8.
28. Thomas D LaToza, Gina Venolia, and Robert DeLine. 2006. Maintaining mental models: a study of developer work habits. In *Proceedings of the 28th international conference on Software engineering*. ACM, 492–501.
29. Timothy C Lethbridge, Susan Elliott Sim, and Janice Singer. 2005. Studying software engineers: Data collection techniques for software field studies. *Empirical software engineering* 10, 3 (2005), 311–341.
30. Yelena Nakhimovsky, Rudy Schusteritsch, and Kerry Rodden. 2006. Scaling the card sort method to over 500 items: restructuring the Google AdWords Help Center. In *CHI'06 Extended Abstracts on Human Factors in Computing Systems*. ACM, 183–188.
31. Seyed Mehdi Nasehi, Jonathan Sillito, Frank Maurer, and Chris Burns. 2012. What makes a good code example?: A study of programming Q&A in StackOverflow. In *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*. IEEE, 25–34.
32. Stephen Oney and Joel Brandt. 2012. Codelets: linking interactive documentation and example code in the editor. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2697–2706.
33. Luca Ponzanelli, Alberto Bacchelli, and Michele Lanza. 2013. Seahawk: Stack overflow in the ide. In *Proceedings of the 2013 International Conference on Software Engineering*. IEEE Press, 1295–1298.
34. Colin Potts. 1993. Software-engineering research revisited. *IEEE software* 10, 5 (1993), 19–28.
35. Lee Rainie and Bernard J Jansen. 2008. Surveys as a complementary method for web log analysis. *Handbook of research on Web log analysis* (2008), 39–64.
36. Yvonne Rogers, Helen Sharp, and Jenny Preece. 2011. *Interaction design: beyond human-computer interaction*. Wiley.
37. Daniel E Rose and Danny Levinson. 2004. Understanding user goals in web search. In *Proceedings of the 13th international conference on World Wide Web*. ACM, 13–19.
38. Helen Sharp, Yvonne Dittrich, and Cleidson de Souza. 2016. The role of ethnographic studies in empirical software engineering. 42, 8 (2016).

39. Jonathan Sillito, Gail C Murphy, and Kris De Volder. 2006. Questions programmers ask during software evolution tasks. In *Proceedings of the 14th ACM SIGSOFT international symposium on Foundations of software engineering*. ACM, 23–34.
40. 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.
41. Janice Singer, Timothy Lethbridge, Norman Vinson, and Nicolas Anquetil. 2010. An examination of software engineering work practices. In *CASCON First Decade High Impact Papers*. IBM Corp., 174–188.
42. Amanda Spink, Bernard J Jansen, Dietmar Wolfram, and Tefko Saracevic. 2002. From e-sex to e-commerce: Web search changes. *Computer* 35, 3 (2002), 107–109.
43. Jeffrey Stylos and Brad A Myers. 2006. Mica: A web-search tool for finding API components and examples. In *Visual Languages and Human-Centric Computing (VL/HCC'06)*. IEEE, 195–202.
44. Christoph Treude, Ohad Barzilay, and Margaret-Anne Storey. 2011. How do programmers ask and answer questions on the web?: Nier track. In *2011 33rd International Conference on Software Engineering (ICSE)*. IEEE, 804–807.
45. Lawrence G Votta. 1994. By the way, has anyone studied any real programmers, yet?[software development process]. In *Software Process Workshop, 1994. Proceedings., Ninth International*. IEEE, 93–95.
46. Li Xiong and Eugene Agichtein. 2007. Towards privacy-preserving query log publishing. In *Query Log Analysis: Social And Technological Challenges. A workshop at the 16th International World Wide Web Conference (WWW 2007)*. May.

## A. PARTICIPANT RECRUITMENT SURVEY

169 responses in total.

I would like to invite you to participate in a study exploring the online search patterns of software engineers. The study involves anonymously tracking the programming-related searches you make online over a period of two weeks. As a thank you for your time, you will be entered into a prize draw to receive one of two £50 Amazon vouchers.

The study is part of my Master’s research project at UCL. If you choose to take part, your data will be treated anonymously and used only for the purpose of this project. No personal details that could be used to identify you will be associated with the data.

Please take 1–2 minutes to respond to a few questions to determine whether you’re eligible to participate. Responding to this survey does not enrol you in the study—if you are eligible, I will provide a detailed information sheet for your consideration.

### 1. Are you actively involved in software development at this time?\*

To answer “Yes”, you should spend a significant amount of your day programming (during most days of the week). This could include any kind of programming—for work, for a side project, as part of a course etc.

- Yes 144 / 85%
- No 25 / 15%

### 2. Do you use Google Chrome as your primary Web browser?\*

- Yes 115 / 80% of 144
- No 29 / 20% of 144

### 3. What is your age?\*

### 4. What is your name?\*

### 5. What is your email address?\*

Thank you for your interest in this study!

You will shortly be sent an invitation to participate. If you have any questions or concerns, contact me at [elise.hein.14@ucl.ac.uk](mailto:elise.hein.14@ucl.ac.uk).

## B. ENROLMENT SURVEY

This is a one-off questionnaire about how you work as a programmer. Please try to complete this questionnaire as soon as possible—preferably on the first day of your participation in the study. It consists mainly of multiple choice questions, and should take you about 10–15 minutes to complete.

You are invited to respond because you agreed to take part in a study exploring developers’ online search habits. If you have any questions or concerns, contact the researcher at [elise.hein.14@ucl.ac.uk](mailto:elise.hein.14@ucl.ac.uk)

Please answer the following questions about yourself.

### 1. What is your gender?\*

- Male 14 / 78%

- Female 4 / 22%

- Other

### 2. What is your age?\*

### 3. What is your nationality?\*

### 4. What is your country of residence?\*

Please answer the following questions about your programming experience.

### 1. How many years of formal training in computer science or a related field do you have?\*

- I am fully self-taught 6 / 33%
- Less than a year 2 / 11%
- 1–2 years 1 / 6%
- 3–5 years 3 / 17%
- More than 5 years 6 / 33%

### 2. Outside of formal training, in what other roles have you used programming?\*

A professional programmer refers to someone who is employed (or self-employed) to produce software.

A professional non-programmer refers to someone who is not employed to produce software, but occasionally does so as part of their job (e.g., data scientist, academic researcher).

A hobbyist refers to someone with no professional programming experience.

Choose as many as you’d like.

- As a professional programmer 17 / 94%
- As a professional non-programmer 4 / 22%
- As a hobbyist 12 / 67%
- Other

### 3. How many years of experience do you have as a [professional programmer | professional non-programmer | hobbyist programmer]?\*

- Less than 1
  - professional programmer 2 / 12% of 17
  - professional non-programmer 0 of 4
  - hobbyist programmer 1 / 8% of 12
- 1–2
  - professional programmer 0 of 17
  - professional non-programmer 0 of 4
  - hobbyist programmer 1 / 8% of 12
- 3–5
  - professional programmer 6 / 35% of 17
  - professional non-programmer 1 / 25% of 4
  - hobbyist programmer 4 / 33% of 12
- 6–10
  - professional programmer 7 / 41% of 17
  - professional non-programmer 1 / 25% of 4
  - hobbyist programmer 1 / 8% of 12
- 11–20
  - professional programmer 1 / 6% of 17
  - professional non-programmer 2 / 50% of 4
  - hobbyist programmer 2 / 17% of 12

- More than 20 professional programmer 1 / 6% of 17
- professional non-programmer 0 of 4
- hobbyist programmer 3 / 25% of 12

**4. Briefly list the main programming technologies you are familiar with.\***

These can include languages, frameworks, platforms as well as development tools. Please don't list technologies that you've used in the past but do not remember in sufficient detail to make use of at the present time.

.....

**5. If there are any other comments you would like to make about your programming experience, please add them here.**

.....

**Please answer the following questions about your current primary programming project.**

Information about your the project you'll be working on during the study will be helpful in analysing your search data. If you're working on multiple projects simultaneously (e.g., work during the day, side-project in the evenings), make your answers about the one that you think you spend more time on.

*Note: Please also make sure that if you've configured a timeframe in the browser extension for this study, it aligns with when you work on your primary project. For example, don't set the plugin to collect data from 10am to 6pm if you work on your primary project in the evenings. You can specify a timeframe from the "Options & info" page (link in plugin popup footer).*

**1. My current primary programming project is ....\***

- A not-for profit project of my own initiative 1 / 6%
- A for-profit project of my own initiative 1 / 6%
- A project I'm employed on as a professional 15 / 83%
- Other 1 / 6%

**2. How many programmers, including yourself, are working together on the project?\***

Please consider only the people who actively contribute to same codebase as you.

- I am the only person working on the codebase 5 / 28%
- 2-4 7 / 39%
- 5-10 3 / 17%
- 11-20 2 / 11%
- More than 20 1 / 6%

**3. Briefly list the technologies you are working with on this project.\***

These can include languages, frameworks, platforms as well as development tools. Please only list the technologies you personally make use of.

.....

**4. Which code editor(s) or IDE(s) do you use when working with these technologies?\***

.....

**5. How would you rate your overall familiarity with the technologies you work with on this project?\***

Please try to assess average or overall familiarity with the parts of the technology stack that you personally need to work with. For example, if you feel very comfortable with Ruby on Rails, and not at all comfortable with CSS, **but must often work with both**, the self-assessment should fall somewhere in the middle.

- 1 (I am not at all familiar with the technologies I work with on this project)
- 2
- 3
- 4 2 / 11%
- 5 6 / 33%
- 6 3 / 17%
- 7 (I am comfortably familiar with the technologies I work with on this project) 7 / 39%

Average: 5.83

**6. How would you rate your overall familiarity with the codebase(s) you contribute to on this project?\***

In this question we are not so much interested in technical expertise, but rather the implicit knowledge about the codebase(s) as a whole: where to add code for new features, which parts need refactoring, what the legacy concerns are, why certain design decisions have been made, etc.

- 1 (I am not at all familiar with the codebase(s) on this project)
- 2 1 / 6%
- 3
- 4 5 / 28%
- 5 5 / 28%
- 6 1 / 6%
- 7 (I am intimately familiar with the codebase(s) on this project) 6 / 33%

Average: 5.28

**7. If there are any other comments you would like to make about your current project, please add them here.**

.....

**Please answer the following questions about how you use the Web when programming.**

**1. Which general search engine do you use for programming queries?\***

"General" here refers to search engines designed for use by the general public. The next question will be about search engines designed specifically for programming queries.

.....

**2. Please list any search engines or other software designed specifically for programming queries that you make frequent use of.**

Examples of such search engines include symbolhound.com or haskell.org/hoogle; examples of software include Dash for offline documentation access. Please only include services where you make explicit use of a search function. For example, only mention Stack Overflow or GitHub if using their search function is a common strategy for you when looking for programming information.

.....  
 3. **If you do make use of any alternative search engines, briefly describe your reasons for preferring them over a general one.**

.....  
 4. **Do you use any plugins or enhancements in your IDE or code editor that provide means of searching the Web or documentation? If so, please describe briefly how they work.**

.....  
 5. **Please describe briefly your usual workflow or set-up for switching between writing code and searching the Web.\***

For example, do you use any keyboard shortcuts for quickly accessing a search engine, or do you keep a dedicated monitor or workspace for the browser only?

6. **Do you use any languages other than English when searching for programming information on the Web?\***

- Yes 2 / 11%
- No 16 / 89%

7. **Please elaborate on your use of languages other than English when searching the Web for programming information.**

What are your reasons for choosing one language over the other?

.....  
 8. **If there are any other comments you would like to make about your search strategies when looking for programming information on the Web, please add them here.**

Thank you for your time!

If you have any questions or concerns, please contact elise.hein.14@ucl.ac.uk.

### C. FOLLOW-UP SURVEY EXAMPLE

This is a quick follow-up survey about three programming-related searches that you made during the 24 hours up to 14:30, Oct 17th. We're interested in understanding the higher-level context around these searches; we're less interested in technical detail.

If you have any questions or concerns, contact the researcher at elise.hein.14@ucl.ac.uk.

**On this day, why did you search for...?**

For each query, please describe, if you can recall, what the trigger was for the search—what problem were you trying to solve? It may be helpful to think of the search engine as a human being. What would your question (or command) have been?

If the query given is not related to programming, please skip the question.

**Example of a useful answer** rails migration string array—I wanted to add a string array to one of my database columns, but I didn't know whether this was possible with rails migrations, or what the syntax is.

1. **ruby encode url params**

This search is from 10:56, Oct 17th.

2. **activerecord map**

This search is from 14:38, Oct 17th.

3. **ruby unix timestamp to date**

This search is from 15:31, Oct 17th.

**On this day, what were you doing when you searched for...?**

We would also like to know about the high-level programming activities you were engaged in when you made each of the three searches. You can choose from the following activities (you'll be able to return to this list for reference).

**Designing** You were analysing a new problem and mapping out the broad flow of code for solving it (e.g., exploratory research for a new feature, API design).

**Building** You were producing code for new program behaviour (e.g., adding a new feature) and getting it into a compatible state.

**Editing** You were editing existing code and returning it to a compatible state (e.g., fixing bugs, refactoring, or rearchitecting).

**Understanding** You were determining information about code such as the inputs and outputs to methods, what the call stack looks like, why the code is doing what it is doing, or the rationale behind a design decision (e.g., investigating a bug, reviewing pull requests).

**Testing** You were creating or running tests, or otherwise ensuring that code is behaving as expected (using a systematic approach outside of your main edit-debug cycle).

**Managing overhead** You were engaged in activities related to the development environment, such as setting up repositories, build or deployment scripts, or managing dependencies.

1. **ruby encode url params**

This search is from 10:56, Oct 17th.

Choose as many as you'd like

- Designing

- Building
- Editing
- Understanding
- Testing
- Managing overhead
- Other

## 2. activerecord map

This search is from 14:38, Oct 17th.

Choose as many as you'd like

- Designing
- Building
- Editing
- Understanding
- Testing
- Managing overhead
- Other

## 3. ruby unix timestamp to date

This search is from 15:31, Oct 17th.

Choose as many as you'd like

- Designing
- Building
- Editing
- Understanding
- Testing
- Managing overhead
- Other

Thank you for your time!

If you have any questions or concerns, please contact [elise.hein.14@ucl.ac.uk](mailto:elise.hein.14@ucl.ac.uk).

## D. EXIT INTERVIEW PROTOCOL

### D.1 Introduction

First of all, I'd like to thank you again for sharing your insights in the study, and thank you for agreeing to this interview. While the insights you've already shared will help to inform the design of better support tools for software engineers, your impressions, frustrations and concerns about how the study was carried out will be valuable in improving this type of data collection method in the future. I thought we might begin with any general impressions you had about the experience. How did you find the two-week data collection period?

### D.2 Questions about Follow-up Surveys

Let's now look at how you found filling in the follow-up surveys on a daily basis.

1. What do you think about the level of commitment required to keep up with the surveys? Was it manageable for you?

*Prompts if required: frustrations with timing; genuine forgetfulness; time it took to respond; whether a routine emerged*

2. You were one of the participants who chose (not) to have daily reminders to respond to the follow-up questions. Do you think this had an impact on when or whether you filled them in?

*Prompts if required: use of the plugin badge; desensitisation to reminders; annoyance with reminders*

3. Could you please describe the experience of filling in a given follow-up survey.

*Prompts if required: difficulties recalling the context of a search; difficulties in deciding depth of annotation; difficulties understanding answer choices; searches that were not related to programming*

### D.3 Questions about Privacy

I'd like to now ask you some questions about privacy issues during the course of the study.

1. Could you please describe how you felt about your privacy during the study?

*Prompts if required: awareness of Incognito mode*

2. Do you think you were aware of enough details about how data was shared in order to make you feel more at ease about taking part in the study?

3. Did you ever forget that your browsing history was being shared? How did this make you feel?

*Prompts if required: regrets about certain browsing activities; frustrations with batches that had already been shared*

4. Because of technical constraints, it was difficult for us to track *only* programming-related browsing. We therefore had to collect all browsing activities. How did you feel about this? Would you say you were ever concerned about any of the programming-related browsing that was shared, or was it only the personal browsing that made you concerned, if at all?

5. Did you ever make use of Incognito mode during the study because you wanted to make sure it was not shared for the research?

6. Did you make use of the two configuration options in the plugin—the timeframe and the blacklist? Why/why not?

*Prompts if required: perceived usefulness; understanding of how configuration options worked; discoverability*

7. Did you ever download a batch of logs from the plugin to review what had been shared? Why/why not?

### D.4 Questions about Search Behaviour

One of the challenges of observing any kind of behaviour in people is that the behaviour observed will sometimes change simply because of the act of observation. In a study like this, we therefore need to find a balance between staying conspicuously in the background, but also making sure you are aware of being watched. I'd like to ask you a few final questions about whether you feel taking part in the study changed your search behaviour in any way at all.

1. During the course of the study, did you ever feel like you altered your search behaviour because you knew your

browsing was being shared—e.g., out of concerns for privacy, or to avoid getting too many follow-up surveys?

2. During the course of the study, did you ever feel like you were more mindful or conscious about your Web search strategies? Do you think this had any effect on how you searched for programming information?

## E. PARTICIPANT CORRESPONDENCE

**Subject:** GDD: Invitation to take part in a study exploring developers' online search behaviour

Hello ... ,

Thank you for your interest in this study!

In the study, we will track participants' programming-related online searches for two weeks using a Chrome extension. I am sending over an information sheet with the full study details for your consideration; please read this carefully. By taking part, you will help us make a small contribution to the design and development of better search tools for programmers! You will also be entered into a prize draw to win one of two £50 Amazon vouchers.

If you would like to participate, respond to this email with the following:

- The date when you would like to begin the study – any day from now up to the 11th of November would be preferred
- Optionally, your telephone number for receiving automated follow-up survey reminder text messages
- A signed copy of the consent form (can be digitally signed)

On the first day of the study, I will send you an email with the Chrome browser extension and setup instructions (installing and configuring the plugin should not take more than a few minutes). At the end of the study I will send you an invitation to take part in an optional exit interview – this is entirely voluntary. The exit interview will be about your participation experience, and will help us to evaluate the design of the plugin and surveys and address any privacy concerns.

I would appreciate if you dropped me a message even if you decide not to take part. If you have any questions, please let me know.

Best wishes,

Elise Hein

**Attachments:** *Information sheet.pdf, Consent form.pdf*

**Figure E.1.** The invitation to participate in the study that was sent to those who indicated their eligibility in the recruitment survey (169 people). The email included copies of the information sheet and informed consent form.

**Subject:** GDD: First day of participation in study

Hello ... ,

Your participation in the two-week study on programmers' online search patterns begins today.

To begin, please install this Chrome extension:

<https://chrome.google.com/webstore/detail/google-driven-development/jjpphncbijckbkpeohmgflobjalohhb>

If the installation is successful, the "Options & info" page for the extension will open automatically. On this page, you can configure the extension to only share data from a specific timeframe, or to completely ignore certain URLs or domains. If you'd like to take advantage of the blacklisting option, I'd recommend taking a quick glance at your browser history during the past few days so that you can identify websites you visit often that are irrelevant to your work as a programmer.

When you click on the extension icon (on the right of the address bar), you will see a link to a one-off survey about your generic development practices. Please try to respond to this at your earliest convenience.

As you have provided me with your mobile number, you'll receive daily automated follow-up reminders at 5pm. Please let me know if this is not a suitable time, or if you would like to not receive these reminders.

The plugin will automatically stop collecting data after two weeks. I'll contact you again at this time with a reminder to uninstall the extension. Meanwhile, if you have any questions or concerns, don't hesitate to get in touch.

Best wishes,

Elise Hein

**Figure E.2.** The welcome briefing sent to participants on the day they wished to begin the study, outlining how to install and configure the browser extension.

**Subject:** GDD: You're now half way through the study

Hello,

You're now half way through the data collection period in the study on developers' online search habits. This is just a short message to check in – let me know if anything is unclear about what is collected, or about the follow-up survey questions that you've been filling in.

I'd kindly ask you to fill in follow-up surveys sooner rather than later, as it becomes harder to recall the context of your searches otherwise. If you haven't yet, please also make sure that you've filled in the generic one-off survey about how you work as a developer – the link to this is also in the plugin popup window.

Many thanks for your continued participation.

Best wishes,

Elise Hein

**Figure E.3.** A check-in message sent to participants midway through the study, notifying them of their progress and reminding them of the surveys that they should fill in.

**Subject:** GDD: Thank you for sharing your insights!

Hello ... ,

Thank you so much for sharing your insights over the past two weeks in our study on how programmers search for information online!

Today is the last day of your two-week data collection period. The browser extension will automatically stop sharing logs, but you may want to uninstall it nonetheless to de-clutter.

The £50 Amazon voucher winners will be drawn once the two-week data collection period has been completed by all who are taking part, at which time I'll send you a separate email letting you know whether you were among the winners.

In the meantime, I'd like to invite you to an optional exit interview to learn about your experience in taking part in the study (I'll not ask you to elaborate on any of your programming searches, nor reveal your participant ID). While the insights you've already shared will help to inform the design of better support tools for software engineers, your impressions, frustrations and concerns about how the study was carried out will be valuable in improving this type of data collection method in the future.

Please let me know if you would be able to take part so that we can arrange a suitable time and place to meet – the interview should take around 15-30 minutes. If you are not in London, we can also chat via Skype or Google Hangouts. Note that you will still be entered into the voucher prize draw whether or not you take part in the interview.

Thank you again for your time and commitment.

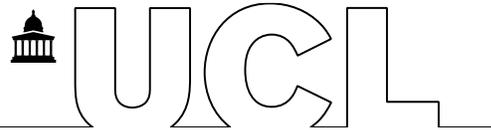
Best wishes,

Elise Hein

**Figure E.4.** The exit briefing sent to participants on the final day of their data collection period, thanking them for their insights and inviting them to an optional exit interview.

## F. INFORMATION SHEET

UCL DIVISION OF PSYCHOLOGY  
AND LANGUAGE SCIENCES



### Information Sheet for Participants in Research Studies

You will be given a copy of this information sheet.

Title of project            **Exploring developers' online search behaviour in the wild**

This study has been approved by the UCL Research Ethics Committee with Project ID Number:

**UCLIC/1617/001/MSc Marquardt/Hein**

#### Investigators

Elise Hein

[elise.hein.14@ucl.ac.uk](mailto:elise.hein.14@ucl.ac.uk)

+44 (0) 758 0426 583

Nicolai Marquardt

[n.marquardt@ucl.ac.uk](mailto:n.marquardt@ucl.ac.uk)

+44 (0) 20 3108 7065 (x57065)

UCL Interaction Centre  
2nd floor 66-72 Gower Street  
London, WC1E 6BT

We would like to invite you to participate in this research study conducted by Elise Hein and Nicolai Marquardt at UCL. You should only participate if you want to; choosing not to take part will not disadvantage you in any way. Before you decide whether you want to take part, it is important for you to read the following information carefully and discuss it with others if you wish. Ask us if there is anything that is not clear or if you would like more information.

One of the challenges programmers face both professionally as well as in their personal projects is being able to find and consolidate relevant information from online sources: tutorials, API documentation, syntax specifications, Q&A websites etc. Though tooling is being constantly developed to bridge the gap between the IDE and the Web browser or to otherwise augment the search experience, few tools have been adopted in industry. In the empirical software engineering community, it is thought that one of the reasons for this may be that design is not rooted in the real work practices of professional software engineers. As such, the purpose of this research project is to gain insight into how real programmers go about searching the Web during their day-to-day programming activities.

In this study, you will be asked to install a browser extension on the computer where you do most of your programming work. The browser extension will automatically collect your browsing data over two weeks and anonymously send it to the research database over a secure connection. Information that could be used to identify you will not be included in the shared data; therefore, the investigators will not be able to associate browsing logs with any one participant. You will also be able to configure the browser extension not to share certain browsing history.

Over the course of the two weeks, the browser extension will also prompt you to respond to questionnaires. The extension will flag one questionnaire at the beginning of the study about your general programming practices, and several short ad-hoc surveys about some of the programming searches that you make during the course of the study (chosen randomly). By describing the context around a given search query, you will help us to understand the search strategies people adopt when faced with different programming challenges.

In total, you should expect to spend about 10-15 minutes setting up the study and responding to the one-off questionnaire, and an average of about 5 minutes per day on subsequent days responding to any surveys that are flagged.

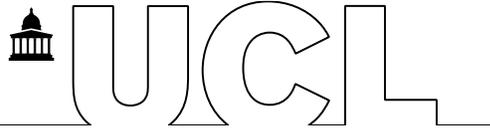
As a final part of the study, you may opt into an exit interview after the two-week data collection period to share with us your experience in taking part in a study of this format. If you opt into the interview, you will **not** be asked about your search history; rather, it will help us to evaluate the design of the study – the browser extension, the surveys, and how to alleviate any privacy concerns.

At the end of the study, all participants will be entered into a prize draw to win one of two £50 Amazon vouchers. A copy of the finished report for this research will also be made available upon request.

All data will be handled according to the Data Protection Act 1998 and will be kept anonymous. Only the investigators on this project will analyze these data. It is up to you to decide whether or not to take part. If you decide to take part you will be given this information sheet to keep and be asked to sign a consent form. If you decide to take part you are still free to withdraw at any time and without giving a reason.

**G. INFORMED CONSENT FORM**

**UCL DIVISION OF PSYCHOLOGY  
AND LANGUAGE SCIENCES**



**Informed Consent Form for Participants in Research Studies**

Title of project            **Exploring developers' online search behaviour in the wild**

This study has been approved by the UCL Research Ethics Committee with Project ID Number:  
**UCLIC/1617/001/MSc Marquart/Hein**

**Participant's Statement**

I .....

agree that I have

- read the information sheet and/or the project has been explained to me orally;
- had the opportunity to ask questions and discuss the study; and
- received satisfactory answers to all my questions or have been advised of an individual to contact for answers to pertinent questions about the research and my rights as a participant and whom to contact in the event of a research-related injury.
- I understand that my browsing history will be partially tracked for two weeks and I am aware of and consent to the analysis of the event logs.
- I understand that my responses in any interviews will be taped and I am aware of and consent to the analysis of the recordings.
- I understand that I must not take part if I am not physically or mentally able to be interviewed.

I understand that I am free to withdraw from the study without penalty if I so wish, and I consent to the processing of my personal information for the purposes of this study only and that it will not be used for any other purpose. I understand that such information will be treated as strictly confidential and handled in accordance with the provisions of the Data Protection Act 1998.

Signed:

Date:

**Investigator's statement**

I, Elise Hein, confirm that I have carefully explained the purpose of the study to the participant and outlined any reasonably foreseeable risks or benefits.

Signed:

Date:

## H. BROWSER PLUGIN WELCOME PAGE



### Google Driven Development

PARTICIPANT ID **PCh1o**

Thank you for taking part in this study exploring the online search patterns of software engineers. For full details about the research, [download the information sheet](#).

This browser extension anonymously shares your browsing data with the researchers for two weeks. Content related to programming will be extracted manually, and all other data will be deleted. Data that can be used to personally identify you (e.g., Google account username, email) will **not** be shared – logs will only be accompanied by a participant ID. Your randomly generated participant ID is **PCh1o**.

The first batch of logs will be uploaded approximately 24 hours after installing the plugin – **Dec 28th 2016, 19:31**. If you don't want any data to be shared at all, please [withdraw from the study](#). Otherwise, the extension will stop sharing data automatically after two weeks – the last batch of logs will be sent on **Jan 10th 2017**.

#### What is shared?

Approximately once per 24 hours, the browser extension will automatically send a batch of logs over a secure connection to the research database. You can see a history of shared batches by clicking on the browser extension icon on the top right of your browser window (  ).

Each batch contains your participant ID and a list of logs. The log items are obtained from Chrome's history API, and correspond to the format specified by `HistoryItem` (see [Chrome History API docs](#)).

#### Timeframe

If there is a significant time of the day when you're not using this computer for programming, you can specify a timeframe where programming-related browsing most likely takes place. When specified, all browsing logs from outside of this timeframe will be omitted from the shared batches.

Share all browsing activity

Only share browsing activity between

09:00 - 18:00

## Blacklisted domains

In order to decrease the volume of shared data, a default list of domains will be omitted from each batch of logs (based on [Top Sites in the UK](#)). You may add your own regular expressions to this list to keep certain logs from being shared. The items you add will **not** be visible to the researchers. **Please don't ignore your search engine, or websites that you may visit when searching for programming information.**

Logs where the URL matches the following regexes will not be shared.

```
file:\\/ /    facebook\\.com    linkedin\\.(co\\.uk|com)    twitter\\.com    reddit\\.com
imgur\\.com    imdb\\.com    theguardian\\.(co\\.uk|com)    bbc\\.(co\\.uk|com)
dailymail\\.co\\.uk    amazon\\.(co\\.uk|com)    paypal\\.(co\\.uk|com)    ebay\\.(co\\.uk|com)
mail\\.google\\.com
```

Note: [regexpal.com](#) is a handy reference. Please don't enter `.*` or other catch-all expressions.

## Privacy concerns

Specifying a timeframe and blacklisting certain domains should help to decrease the proportion of logs not related to programming in shared batches. We also manually go through each batch and delete everything we deem irrelevant to programming. If you want further control over what logs are shared, we encourage you to make use of Incognito mode for browsing that you want to keep private. Finally, any items that you delete from your browser history before a batch is sent will also not be shared.

## Follow-up questions

For each batch of logs, we may flag certain search queries you've made for a follow-up – this is what allows us to better determine programmers' goals when they make a specific search.

If a follow-up survey is available, it will show up next to its batch when you click on the extension icon. We encourage you to respond as soon as possible, when it's still easy to recall the context of the search. A follow-up survey should not take more than a few minutes to fill in.

## Withdrawing from the study

You can withdraw from the study at any time. To do so, please uninstall this browser extension. Any logs that will have already been shared will be deleted, and will not be made use of in the research.

For any questions or concerns, please contact [elise.hein.14@ucl.ac.uk](mailto:elise.hein.14@ucl.ac.uk).

Figure H.1. The welcome page displayed to participants when they installed the browser extension. Note: Figure spans two pages.



## J. BATCHES PER PARTICIPANT

|                | <b>Total batches</b> | <b>Batches with published survey</b> | <b>Ratio of surveys to batches</b> |
|----------------|----------------------|--------------------------------------|------------------------------------|
| P3             | 9                    | 9                                    | 100%                               |
| P6             | 13                   | 12                                   | 92%                                |
| P14            | 10                   | 8                                    | 80%                                |
| P4             | 13                   | 10                                   | 77%                                |
| P10            | 10                   | 7                                    | 70%                                |
| P17            | 10                   | 7                                    | 70%                                |
| P2             | 15                   | 10                                   | 67%                                |
| P12            | 12                   | 8                                    | 67%                                |
| P1             | 12                   | 8                                    | 67%                                |
| P15            | 6                    | 4                                    | 67%                                |
| P9             | 13                   | 7                                    | 54%                                |
| P18            | 10                   | 5                                    | 50%                                |
| P11            | 13                   | 6                                    | 46%                                |
| P16            | 9                    | 4                                    | 44%                                |
| P5             | 11                   | 4                                    | 36%                                |
| P8             | 11                   | 4                                    | 36%                                |
| P7             | 10                   | 3                                    | 30%                                |
| P13            | 14                   | 3                                    | 21%                                |
| <b>Total</b>   | <b>201</b>           | <b>119</b>                           |                                    |
| <b>Average</b> | <b>11.2</b>          | <b>6.61</b>                          | <b>59.69%</b>                      |

**Table J.1.** The number of batches shared by each participant, and the number of batches with a published follow-up survey. Recall that a follow-up was only published if there were at least three occurrences of programming searches from separate search sessions in the batch.