

Archetypal Internet-Scale Source Code Searching

Medha Umarji
*Department of Information
Systems
University of Maryland,
Baltimore County
medha1@umbc.edu*

Susan Elliott Sim
*Department of Informatics
University of California, Irvine
ses@ics.uci.edu*

Cristina V. Lopes
*Department of Informatics
University of California, Irvine
lopes@ics.uci.edu*

Abstract

In recent years, a number of search engines have appeared on the Internet for the purpose of searching repositories consisting of hundreds of millions of lines of Open Source code. However, it is not clear precisely what, how, and why programmers search for code on the Internet. To this end, we conducted a web-based survey to understand the source code searching behavior of programmers, specifically, their search motivations, search targets, tools used, and code selection criteria. Data was collected from 69 respondents, including 58 specific examples of searches. We applied open coding to these anecdotes and found two main search motivations, as well as, a range of sizes for search targets. The first motivation was searching for source code that could be dropped into a project and used immediately. The second motivation was searching for reference examples of source code to provide. The targets of these searches could range in size from a block (a few lines of code) to a subsystem (e.g. library or API), to an entire system. Using combinations of motivations and target sizes, nine archetypes were identified. As well, the overall search process is discussed. The paper concludes with some guidance on how to use these archetypes to create personas for the purpose of designing Internet-scale source code search engines and for generating scenarios for evaluating these tools.

1. Introduction

With the increasing popularity of Open Source, a rapidly growing quantity of source code is available on the Internet. Software developers are now using this rich resource in their work. Evidence of this practice can be found the number of project hosting sites, code repositories, and source code search engines that have appeared. Among these are Koders.com with over 226 million lines of code (MLOC), Krugle.com with over 2 billion lines of code (TLOC), csourcesearch.net over 283

MLOC, and Google Code Search with over 1TLOC. These source code search engines treat Internet-scale code searching in much the same manner as code search within a single project in an integrated development environment. However, there has been little research to support this assumption.

This study was conducted to characterize how Internet-scale source code searching—What do developers look for? How do they find what they are looking for? What tools do they use? When do they decide to search? To this end, we conducted a questionnaire-based survey of software developers contacted using availability sampling over the Internet. The design of this study is based on previous surveys by Eisenstadt [1], and Sim, Clarke, and Holt [2]. Using an online questionnaire, we collected data from 69 respondents who were solicited using newsgroups and mailing lists.

Their responses and anecdotes were analyzed systematically to find common themes, or archetypes. An archetype is a concept from literary theory. It serves to unify recurring images across literary works with a similar structure. In the context of source code searching, an archetype is a theory to unify and integrate typical or recurring searches. As with literature, a set of them will be necessary to characterize the range of searching anecdotes.

We found that there are two major search motivations. The first motivation was searching for piece of code that can be reused, for example, a text search engine like Lucene, or a graphical user interface (GUI) widget. The second motivation was searching for a reference example. In effect, developers were using the World Wide Web as a very large desk reference. Search targets that varied in size and could be small-grained, such as a block of code, medium-grained, such as a package, or large-grained, such as an entire system.

These archetypes can be used in two ways: to design Internet-scale source code searching tools; and to evaluate these same tools. For the purpose of design, the anecdotes and archetypes can be used to create personas, a set of

representative user profiles. For the purpose of evaluation, the results can be used to generate scenarios that represent typical searchers.

The remainder of the paper is organized as follows. Section 2 reviews related work on source code searching, code reuse, and Open Source. The design of the study is described in Section 3. Results of the study are presented in Section 4. The archetypes are presented in Section 5. Section 6 highlights some of the issues in the search process. Section 7 discusses how the anecdotes and archetypes can be used to create personas and scenarios to design and evaluate code search engines. Future work and concluding remarks are presented in Section 8.

2. Related Work

The work in this paper has evolved from past research and current trends in software development. The two trends that motivate this research are the increasing availability of source code on the Internet, and the emergence of tools for accessing the source code. The source code available on the web comes from Open Source projects, web sites that support communities of practice, and language-specific archives. Collectively, these sites contain billions of lines of code in countless languages. As is the case with web pages, it can be difficult to locate a particular resource. General-purpose search engines, such as Google and Yahoo!, can be used, but they don't take advantage of structural information in the code. To fill this need, code-specific search engines have been created. These software tools leverage the technology and know-how from source code searching tools within programming environments. However, code search on the Internet at times is more similar to code reuse than the find function in an IDE. In this section, we will review the trends and results that motivate and inform our research.

2.1 Source Code on the Internet

The Open Source movement has dramatically increased the quantity of source code available on the Internet. While the Open Source concept has been around for decades, it is only in the last ten years or so that it has become commonplace. For-profit corporations are now contributing source code and person-hours to Open Source projects [3]. In addition to the applications themselves, the source code provides "rich base of reusable software" [4]. Sourceforge.net and freshmeat.net host thousands of projects and have an infrastructure that supports the sharing of programs and knowledge.

In addition to Open Source projects, communities of practice have created collections of source code. A community of practice is formed by a group of people united by a joint enterprise, who develop mutually beneficial social relationships in the process of working towards things that matter to them [5]. In software

development, communities of practice have evolved around programming languages, platforms, applications, and software tools through technology-specific mailing lists, blogs, and wikis. For example php.net is a comprehensive reference for the PHP language with places for community members to add advice. On these lists and sites, people not only contribute not only source code, but also contributions of helpful tips on what works, what doesn't, and what is the best way to solve a certain problem.

Language-specific repositories are another source of code on the Internet. Certain languages have web sites that allow users to share components or widgets with each other. Examples of these are the Comprehensive Perl Archive Network (CPAN.org) and Java Source and Support (java2s.com).

2.2 Code Search Engines

General-purpose search engines such as Google and Yahoo! are used for code search most often. Users are familiar with these tools and due to their effectiveness in retrieving documents on the web they are easily the most popular. They are effective in broad searches for functionality, when good search terms are available. However, these search engines are not able to exploit the context and relationships between source code snippets, as they treat source code like a bag of words.

Code-specific search engines crawl the web to create a repository of publicly accessible code and enable users to search based on a variety of attributes such as identifiers, programming languages, code licenses, and platforms. While the search is limited to the repository, the amount of code available is huge, many millions of lines of code or classes. Three of the major code-specific search engines are Krugle, Koders, and Google Code Search. It also allows users to explicitly search for class, function, and interface definitions using regular expressions. Krugle returns links not only to source code, but also to tech pages, books and projects. It has a good visualization for browsing code repositories and also supports tab-based searching. The searches can be applied to different segments: source code, comments, and definitions (class or method). Google Code Search includes support for regular expressions, search within file names and packages, and case-sensitive search. Sourcerer, a research prototype, is experimenting with new approaches to code search, such as applying the page rank algorithm to call graphs and code fingerprints as a search term. [6].

A problem with all of these tools is they do not support the social interaction processes that are the lifeline of any project. Neither do they support the formation and sustenance of communities of practice that are so essential for learning and sharing in any domain [5].

2.3 Source Code Searching

Source code searching within development environments has been around for many years. There have been studies of how and why programmers search, as well as, advances in searches tools.

In a study of software engineering work practices, Singer et al. [10] found that searching was the most common activity for software engineers. They were typically locating a bug or a problem, finding ways to fix it and then evaluating the impact on other segments. Program comprehension, reuse and bug fixing were cited as the chief motivations for source code searching in that study. A related study on source code searching by Sim, Clarke, and Holt [2] found that the search goals cited frequently by developers were code reuse, defect repair, program understanding, feature addition and impact analysis. They found that programmers were most frequently looking for function definitions, variable definitions, all uses of a function and all uses of a variable.

Source code searching began as simple matching of search terms within an editor. Regular expression soon followed, as did the Unix-based grep facility [7], the find command in Unix and also the File Find command under Microsoft Windows [2]. Modern source code searching tools, such as those in integrated development environments, make use of program language syntax, parse trees, and even architectural information. As well there are research tools that perform concept assignment [8] and feature identification [9].

However, none of these existing tools have capabilities to search for software components based on functionality. These are features more commonly discussed in the context of tools to support code reuse.

2.4 Software Reuse

Source code searching on the Internet has more commonalities with the phenomenon of software reuse, than with traditional source code searching for program understanding and bug fixing. Programmers are looking more for a component to reuse than to find where a variable is declared.

Reuse is a common motivation for internet-scale source code searching. Spinellis and Szyperski wrote, "...reuse granularity is not restricted by the artificial product boundaries of components distributed in binary form (which marketing considerations often impose). When reusing open source, code adoption can happen at the level of a few lines of code, a method, a class, a library, a component, a tool, or a complete system" [4].

Research in reuse in proprietary settings involves indexing and storing software components in a way that would make retrieval and usage easy [11]. An assumption in traditional software reuse is the component is expected to be used without modification. Modification is an

expensive operation; making non-trivial changes quickly increases the effort of understanding a component and any savings over implementation from scratch is lost. Reuse of Open Source code occurs with an understanding that effort will be expended in contextualizing, comprehending and modifying the code. Another difference that in corporate settings, reuse is planned and components are created for that purpose. On the Internet, most of the code has not been designed for reuse.

One of the challenges in code reuse is aligning the vocabulary for specifying searches with the search targets. Fischer et al. [12] discussed the gap between the system model of the software and the user's situation model, which makes it difficult for the user to express a requirement in a language that the system can understand. The problem of discourse persists in Internet-scale search as the primary method of searching continues to be keywords and regular expressions.

3. Method

Online surveys have become increasingly common over the last decade, as Internet usage has grown by leaps and bounds. They have become an established empirical method, especially for Internet usage [13].

Studies have been conducted to improve understanding of why users look for information, their search requirements, their search strategies, backgrounds and experiences, and their comparative assessment of available search mechanisms [14] [15]. Sim, Clarke, and Holt [2] conducted an online survey in late 1997 of source code searching among programmers that served as the model for this research. Underpinning these research designs are traditional survey methods that have been used in the social sciences for many years [16].

3.1 Research Questions

In this study, we wanted to gain an understanding of how software developers currently search for source code on the Internet. The search features on project hosting sites and the emergence of source code-specific search engines hint at the kinds of search taking place, but empirical data is needed. Therefore the research questions for this study were as follows.

- What tools do programmers use to search for source code on the Internet?
- What do they look for when they are searching source code?
- How do they use the source code that is found?

Data on what tools are used provide information about the skills and tendencies of programmers when searching the web. The search targets and usage patterns for the code suggest new features. Answers to the last two questions were obtained from the open ended questions, when analyzed resulted in search archetypes.

3.2 Data Gathering Instrument

We designed an online survey with 13 closed-end questions and two open-ended questions (see Appendix A). Design of the questionnaire began with interviews with programmers and continued with a small trial of an initial version of the questionnaire.

The survey had questions about i) types of information sources used by programmers while searching, ii) popular search sites iii) filtering criteria used by programmers, and iv) programming languages used. Background information was also solicited, to help us understand the different types of users and their experiences. One of the open-ended questions asked respondents to detail a scenario when they were searching for source code on the Internet, and include as much context and problem-related information as possible.

3.3 Define the Population and Sampling Method

Our goal was to cover a wide range of people that search for source code often, to get a representative sample. The population was any programmer who had searched for source code on the Internet. However, it was not possible to obtain a systematically random sample, and availability sampling also known as convenience sampling was the chosen sampling technique.

Convenience sampling may pose a threat to external validity of the results. However, this was an exploratory study and the goal was to collect data on a variety of behavior, and not its prevalence, so availability sampling was considered adequate for this task.

We solicited participants from a number of mailing lists and newsgroups. We attempted to solicit participants through Open Source news web sites, but were declined. This strategy gave us access to a large number of developers and users of open source software, as well as developers who worked on proprietary and commercial software.

3.4 Administer the Survey

Invitations to participate in the survey were posted to the following newsgroups and mailing lists: Javaworld mailing list, the cgi beginner's list on perl.org, comp.software-engg, comp.lang.c, and comp.lang.java. The survey was open for six months to collect responses.

3.5 Analyze the Data

The data was analyzed using a combination of quantitative and qualitative techniques. The multiple-choice questions were coded using nominal and ordinal scale variables. For the open ended questions, the responses were text descriptions that were analyzed qualitatively. We analyzed them for recurring patterns using open coding [17] and a grounded theory approach [18]. Without making prior assumptions about what we would find, we developed codes for categories iteratively

and inductively. The two authors analyzed the data separately, and we found a high level of agreement in our categories. Subsequently, we combined our codes and refined the categories for clarity of presentation.

3.6 Threats to Validity

The main shortcoming of this study is generalizability, i.e. the sample of respondents is not sufficiently representative of the population. This is basic problem with empirical research in software engineering is there is not a reliable model of population characteristics so that the representativeness of a sample can be assessed. This study is no exception. Furthermore, we only solicited participants from mailing lists and newsgroups. Therefore, we are not trying to quantify the prevalence of certain types of behavior, nor are we using inferential statistics. Instead, we are looking for a variety of search behaviors and patterns (or archetypes), which is appropriate for an exploratory study.

4. Results

A total of 69 people responded to the survey and provided descriptions of 58 situations where they searched for source code on the Internet. The quality of the responses varied greatly. Some respondents only filled in the multiple-choice questions. Others provided very terse descriptions of search situations. Yet others provided extremely detailed descriptions of more than one situation.

A majority of the developers that responded to our survey were programmers in Java (54), C++ (58) and Perl (32). More than half of the respondents (37) had contributed to an Open Source project. Most of the respondents (41) had experience working on small teams with 1 to 5 people.

4.1 Situations

We analyzed 58 scenarios of source code searching. They ranged in length from one to ten lines. Figure 1 below is an example of a typical response that we received.

```
Sometimes I did a source code
searching when I don't know how to use
a class or a library. For an example I
didn't know how to create a window
using SWT class. I did a Google search
with the description of what I want to
do. I decided on the best match based
on whether I understand the example
code.
```

Figure 1: Example Search Description

The anecdotes were categorized among a number of dimensions. Clear patterns emerged regarding two aspects of their searches: i) what programmers were searching for; and ii) how they searched for it.

In terms of what programmers were searching for, anecdotes were categorized along two orthogonal dimensions: the motivation for the search and the size of the search target. Some responses had multiple search targets and motivations, and in such cases, each was coded separately. The most specific code that was appropriate for the search was selected, based on the information given by the participant. In **Error! Reference source not found.**, the motivation was coded as “reference example” and the size of search target was coded as “subsystem”.

	Code for Reuse	Reference Example	Row Total
Block	8	4	12
Subsystem	21	11	32
System	5	2	7
Column Total	37	22	51

Table 1: Purpose by Target Size

4.2 Motivations for Searching.

Detailed analysis of scenarios showed that respondents were either searching for reusable code (37) or reference examples (22). Reusable code is source code that they could just drop into their program, such as an implementation of trie tree data structure, quick sort algorithm, and two-way hash table. A reference example is a piece of code that showed how to do something, for instance, how to use a particular GUI widget, what is the syntax of a particular command in Java. Searches of this type essentially use the web as a large desk reference. These two motivations were emerged very clearly in the anecdotes, and almost all the scenarios could be neatly classified into either of these two motivations.

A key difference between the two motivations is the amount of modification that searcher intended to perform. Programmers seeking reusable code planned to find pieces that could be dropped into a project and used right away. For example, “Needed to convert uploaded images of all types into jpeg and then [generate] thumbnails. Due to timescales; it could not be done in house...” Those seeking reference examples intended to re-implement or significantly modify the code found. One respondent wrote, “I typically search for examples of how to do things; rather than code to use directly. The products that I work on are closed-source, one can’t [use] most open source directly.”

On occasion, the search was initially seeking reusable code would fail and become a search for reference information. A programmer needed a mutable string class in Java, but the results from search engines either had only a minimal implementation or an inappropriate Open Source license. He wrote, “...in the end I just rolled my own,” using the other implementations for ideas.

4.3 Size of Search Targets.

Across both types of searches, the size of the search target varied in a similar fashion. The sizes of the search targets were classified as a block (12), a subsystem (32), or a system (7). A block was a few lines of code, up to an entire function in size.

Common block-sized targets were wrappers and parsers (3), and code excerpts (8). A number of the searches for code excerpts were for PHP and JavaScript. Programs in these languages tended to be small and plentiful, which meant it was easier to make use of a few lines of code. There were searches for a small section of code that solved a specific problem, such as “encode/decode a URL” and “RSS feed parser”.

A subsystem was piece of functionality that was not a stand-alone application, and the programmer searching intended to use it as a component. Categories of subsystem targets are implementations of well-known algorithms and data structures (14), GUI widgets (9), and uses of language features (6). Some examples from the data include “a Java implementation of statistical techniques like t-test” and “wrapper code for the native pcap library.”

A system was an application that could be used in its own. Searchers often intended to use these as a component in their own software. Respondents were “looking for some big piece of code that would more or less do what I want...” or something that would show them “how to do it.”

The results are presented as archetypal searches in Section 5 and as observations about the search process in Section **Error! Reference source not found.**

5. Archetypal Searches

By examining the motivations for search and the size of search targets, we found common or more frequent relationships. These patterns, or archetypes, are presented in this section, as well as, some unusual, but interesting searches.

5.1 Common Searches

The most common type of search was a subsystem that could be reused. Archetypes 1, 3, and 4 fell into this category. The next most common search, archetype 2, was for a system that could be modified or extended to satisfy the needs of the project. Archetypes 5-8 are searches for examples of how to do something, such as using a component or implementing a solution. The final archetype is searching for reports and patches for bugs.

1. Reusable data structures and algorithms to be incorporated into an implementation.

Eight of the reported searches were for algorithms and data structures, such as “two-way hash tables,” “B+ trees,” “trie trees,” and “binary search algorithm.” were

included at this level of granularity. We suspect that this was the most prevalent because there was a close match between the vocabulary for describing the object in code and the vocabulary for describing the search. Furthermore, abstract data structures are a well-understood basic building block in computer science.

2. A reusable system to be used as a starting point for an implementation.

While creating a new system, developers often look for systems that they can use as a starting point. There were seven such searches by developers who were looking for “stand-alone tools” or a “backbone for an upcoming project” or just a “big piece of code that does more or less what I want”. Examples of search targets included an application server, an ERP package or a database system. We conjecture that this type of search is common because a system does not need to be de-contextualized before it is used in a new project. Also, systems are easy to find because they typically have web sites or project pages that contain text descriptions of the software. Finally, customizing an existing application saves a lot of time in comparison to implementing from scratch.

3. A reusable GUI widget to be used in an implementation.

Developers often looked for widgets for graphical user interfaces and there were seven searches of this kind in our data. Users searched by keywords of the functionality that they desired, for example – “inserting a browser in a Java Swing frame”. Searches for functionality are somewhat independent of the source code implementation underneath, but are mainly concerned with feature addition. Other examples include a “Java interface for subversion” and a source code that creates a “SeeSoft-like visualization”. We believe that searches for GUI widgets are popular, because these components are easy to reuse. A software developer need only ensure that the widget is compatible with the GUI framework being used on the project. As well, GUI widgets can be displayed visually, therefore, making it easier for a developer to quickly assess the appropriateness of the search result.

4. A reusable library to be incorporated into an implementation.

There were six searches for a reusable library, sometimes called a package or API. Programmers were generally looking for a subsystem that could be dropped in and used immediately. Some examples of the searches were for “speech processing toolkits”, “library for date manipulation in Perl” and “Java implementations of statistical techniques.”

5. Example of how to implement a data structure or algorithm

In six instances, developers looked for source code snippets to verify their solution or to aid reimplementing, e.g. “to verify the implementation of a

well-known algorithm.” There were six searches were for a piece of code to use as a reference in order to develop the same functionality. An implementation was more informative than a description or pseudocode, because the implementation had been test and could execute. Respondents believed that a reference example would show them the right way to do something, and a running program had a lot of credibility.

6. Example of how to use a library.

Developers looked for examples of how to use a library, for instance, “Sometimes; I did a source code searching when I don't know how to use a class or a library.” There were six anecdotes reporting this kind of search. Libraries and APIs can be complex to use, with arcane incantations for calling methods or instantiating classes. A reference example is easier to use than documentation, because it gives the programmer a starting point that can be tweaked to suit the situation.

7. Example of how to use features of a programming language.

In four anecdotes, respondents reported that reference examples of language syntax and idioms were helpful when working with an unfamiliar programming language. Users who haven't programmed in a language before, or have forgotten parts of it, or are using the language in a new way (e.g. new hardware), searched for source code to serve as a language reference. One respondent wrote, “...mostly I look for code for syntax, I don't always like to refer to books for syntax if it is readily available on my desktop.”

8. A block of code to be used as an example.

Developers look at a block of code to learn how to do something. There were four situations that described this type of search. Programmers were not looking for reusable components, but their goal was to learn through examples, such as “examples of javascript implementation of menus” and “examples of thread implementation in python”.

9. Confirmation and resolution of bugs

There were five searches that were looking for solutions to a bug. Sometimes the solution can in the form of a report or post to a forum that confirmed the presence of an actual bug. At other times, there was a patch that repaired the bug. Three of the searches led developers to find relevant information in mailing lists and forums. Developers prefer to search for a patch or quick-fix by forming natural language queries with the keywords from an error message or keywords based on the functionality deviation caused by a bug. The need for code in such situations is very specific in terms of implementation language, platform, version information, size of patch and licensing requirements. In the process of debugging, if the problem seems to occur while compiling a library or at

run-time, users examine the source code of a library to determine the exact problem.

5.2 Uncommon Searches

There were a few uncommon searches that are worthy of attention. These were looking for a system to be used as a reference example, seeking a reference example for using a GUI framework or widget, and searching for examples of language syntax usage.

While developing or modifying a system, programmers look at existing similar systems for ideas. Searches for systems that can be understood and the logic/principles can be borrowed to construct new systems. Two searches were for systems with similar functionality as the current or to-be system. This technique was mainly used in a proprietary environment or when it was easier to construct a new system rather than adopt an existing one.

There were also two searches for examples of how to use GUI widgets. These included searches for code samples on how to use a particular component from Swing and Microsoft Foundation Classes.

Finally, there was one anecdote from a programming language designer who searched to find examples “in the wild” of syntax from the language. This information was used to evaluate requests for changes and suggestions for features.

6. Process of Search

In this section, we discuss the process that respondents used to search for source code. In particular, we examine how they began the search, the tools that they used, and the criteria for making the final selection. There are two themes that emerge in this analysis: i) the importance of social interaction in the process; and ii) the difficulty of finding appropriate search terms for source code.

6.1 Starting Points

A search for source code on the Internet begins long before a programmer sits down in front of a computer and selects a web site. The starting point for a search relies extensively on background knowledge, obtained by following Open Source news, reading news and articles, conversations with friends. Searches often begin with a peer recommendation (30), sometimes solicited, sometimes exchanged in casual conversation. These advice and opinions were highly valued, especially if the person has used the software. For instance, one respondent stated, “...friends recommended Graph.pm searched for that on search.cpan.org, and found it was just what I needed.”

6.2 Tools and Information Sources

Typically, the first tool that programmers turned to for searching out source code was a general-purpose search

engine, such as Google or Yahoo!. In answer to a multiple choice question about resources used in search, we found that 60 of the 69 respondents did so. In this question, respondents were asked to select all options that applied. Project hosting sites came next, with 34 of the respondents using them for source code search.

	Count
Google, Yahoo, MSN Search etc	60
Domain knowledge	37
Sourceforge.net, freshmeat.net	34
References from peers	30
Mailing lists	16
Code-specific search engines	11

Table 2 Tools and Information Sources

A recommendation and a good text search engine is a powerful combination. The recommendation usually names a system, which can then be used as a search term. Domain knowledge can also provide the name of a system, which can then be used on a project hosting site. The name of a system plus a little context coaxes good results even from search engines not designed to search source code. However, this shortcoming is not a real difficulty, because programmers are looking for functionality, not elements in the source code.

Mailing lists (16) also provided good information. On these lists, newsgroups, and online forums, other programmers talked about source code. Consequently, natural language vocabulary becomes associated with a particular piece of software by proximity. Once again, it is a work-around the problem of specifying functionality using code elements. As one developer commented, “In general the sort of question I have is often asked and can be easily found by searching google for the question in general (in natural language) in the hopes that the question has been asked on a forum somewhere and a response from someone experienced has been posted.”

Only 11 out of 69 respondents reported using a code-specific search engine. Some were very skeptical. One programmer wrote, “In short, I would never rely on a ‘code search engine’. This idea is just plain silly. Sort of ivory tower. If you want to find something usable you have to look for ‘people already using it.’”

Once users have access to the source code that matches their requirements, the problem of narrowing down the list of retrieved items arises.

6.3 Selection Criteria

A search engine typically returns a very large number of hits. Consequently, searchers need to decide which hits to inspect and ultimately use. To our surprise, code quality was never mentioned as a criterion. The most important single criterion was functionality. All other considerations were secondary. This makes sense,

because once the requisite functionality was found, the choice was usually limited.

Table 3 summarizes the responses to a multiple-choice question regarding selection criteria. Respondents were permitted to select more than one option.

Criteria	Count
Available functionality	54
Terms of license	30
Price	26
User support available	21
Level of project activity	18

Table 3 Criteria for selecting a code component

The software license was second most important. We were somewhat surprised by this finding, but suspect that respondents were distinguishing between Open Source and commercial products, as opposed to different Open Source licenses.

The anecdotes repeatedly emphasized the ease of use and the availability of a community of practice as important factors in making a selection. The software had to be easy to install and have good documentation. As one user put it, "...[the core developer] had good documentation of his code with lots of comments too (by which I could also modify his code), hence I decided to use that code." A piece of software with the required functionality may be eliminated if the user can't easily determined whether it has the required features.

While the respondents did not use the term community of practice, they often referred to its elements. They preferred to use code that came with an active group of developers and users. These people, whether they were local peers or Internet acquaintances, could be approached for help,

Help from a local expert, an electronic forum, a mailing list archive, or active users who are doing similar tasks and are willing to answer questions is a major consideration while choosing Open Source software. One respondent said he looked for "... issues which are then found by people, solved and posted on mailing lists of discussion forums." A glance at the forums tells the users how friendly a project is and how likely they are to obtain help when needed. Another developer wrote, "The criteria that I used were: 1) if the tool was in java 2) if the tool was web based 3) the activity of the project." Activity can be quantified as the number of contributors, frequency of builds and updates, traffic on newsgroups, and number of users. Larger projects have more resources, are more responsive, and are more likely to rank highly on these criteria.

Despite the fact that programmers were geographically dispersed, there was a real yearning for community. One respondent summarized the search process this way,

"So the key is to find a community or association or contact team in [a] partner company who is working on somewhat structural similar problems. Then you have to 'hang out' and see what people are doing to solve their problems. Finally, if you have some candidates always look at the bug tracker and support forums in order to see **what sort of problems** people have using it."

Overall, social characteristics of the project, such as the level of activity, presence of discussion forums, and recommendations by peers seem to have precedence over characteristics of the source code, such as whether the code is peer reviewed and tested.

7. Applications of Archetypes

Search archetypes provide valuable insights into how programmers search for code on the Internet. They can be used in both the design and evaluation of search and other tools. One way that the archetypes can be used in design is by creating personas to represent typical users. Archetypes can also be used in tool evaluation by creating typical usage scenarios. In this section, we present examples of these applications.

7.1 Personas for Design

Personas are depictions of fictitious people with realistic characteristics. These "characters" are given lives as people, not just as software users—they have jobs, a car and a family. When used to represent typical customers or users, these personas are powerful because they contain enough technical details to inform a design and enough personality that designers relate with them. Consequently, designers are motivated to create a high quality product that comes as close to real needs and real uses as possible. Often a set of personas is needed to represent the full spectrum of potential users. Care must be taken to craft these personas so that they capture an appropriate set of characteristics. Designers must also avoid turning personas into caricatures, straw men, or hollow shells.

For best results, personas should be based on real users and real examples of usage. These examples can be obtained through online surveys, contextual interviews and focus groups [19]. An excerpt of an example persona is presented below.

Andrew is a 42-year-old software developer working for works for a software company that specializes in enterprise and back office applications. He has been married for 3 years and has no children. He and his wife live in a downtown apartment and his job is a short commute away by public transit. He likes sushi and stays active by jogging and playing volleyball....

7.2 Scenarios for Evaluation

The archetypes can also be used to create scenarios that can be used to evaluate tools for search. A scenario describes a particular situation where a search for source code is being performed on the Internet. Whereas archetypes are abstract, scenarios describe a specific instance of a class. As well, scenarios are more concrete and include context.

A scenario created for the purpose of evaluating search tools should include sufficient information for an unbiased observer or subject in an experiment to make a judgment about the relevance of hit returned by a search engine. Relevance judgments are needed to calculate precision and recall, two standard metrics from information retrieval for evaluating search engines [20]. Usage scenarios provide a balance between contextual information and objective criteria.

A scenario should include the following information: i) goal of search (component or example); ii) size of search target (block, subsystem, or system); iii) programming language; iv) context of search; and v) a situation linking the first four parts. All of this information is necessary to judge the relevance of an item returned by a search engine. The first two parts of the scenario are based on the archetypes from our study. Programming language is included because code in some languages is easier to find than others, e.g. PHP vs. C. As well, the context affects the usefulness of items returned, i.e. academic vs. industrial settings, Eclipse plug-ins vs. web sites. An example of a usage scenario is presented below.

You are working in the Python programming language, and need to have multi-threading functionality in your program. You have never used threads before, and would like to know how to create threads, to switch between threads, and so on. Look for examples by which you can learn. Any thread implementations of Python programs are relevant. Remember you will not be using the code directly, you will like to learn how to use it.

We are currently conducting an empirical evaluation of different Internet-scale source code search engines using scenarios. To ensure that each abstract archetype is covered appropriately, we have generated multiple scenarios for each one, which are assigned to subjects randomly.

8. Conclusions

The goal of this research study was to gain an understanding of Internet-scale source code. To this end, we conducted an online survey of 69 programmers and collected 58 examples of searches. The data indicated that programmers mainly searched for either reusable code or reference examples. The size of search targets ranged from a block of code to a subsystem to an entire system. The contribution of this work is insight into the variety of

Internet-scale source code searches that programmers perform. Until now, there has been little data what programmers looked for, how they found it, and how they used it.

Developers not only look for code, they also look for a community of practice, that is, a social group where they share knowledge and expertise. Domain knowledge and communities of practice are critical parts of the search process. In addition to search engines, programmers reported that they use project hosting sites and social tagging websites to find code.

The results of this study are applied in the formation of personas to aid the design of search tools, and in the formation of scenarios for evaluating existing search tools. Tools for locating source code would be complemented by recommender systems and reviews; more like Amazon.com and less like a library catalog. One of the future directions for this line of research is the use of scenarios to evaluate different search engines. We are continuing with experiments in this vein and hope to continue making contributions to Internet-scale source code searching.

9. Acknowledgements

Thanks to Crista Lopes who supported this research, and provided encouragement and early feedback.

References

- [1] M. Eisenstadt., "My hairiest bug war stories," *Communications of the ACM*, vol. 40, pp. 30-37, 1997.
- [2] S. E. Sim, C. L. A. Clarke, and R. C. Holt, "Archetypal source code searches: A survey of software developers and maintainers " in *Proceedings of the 6th International Workshop on Program Comprehension IEEE Computer Society*, 1998 pp. 180
- [3] R. Goldman and R. P. Gabriel, *Innovation happens elsewhere: Open source as business strategy*. San Francisco, CA: Morgan Kaufmann Publishers, 2005.
- [4] D. Spinellis and C. Szyperski, "Guest editors' introduction: How is open source affecting software development? ," *IEEE Software* vol. 21 pp. 28-33 2004
- [5] J. L. E. Wenger, *Situated learning: Legitimate peripheral participation*. England: Cambridge University Press, 1991.
- [6] B. Sushil, N. Trung, L. Erik, D. Yimeng, R. Paul, B. Pierre, and L. Cristina, "Sourcerer: A search engine for open source code supporting structure-based search," in *Companion to the 21st ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications*. Portland, Oregon, USA: ACM Press, 2006.
- [7] J. Singer and T. Lethbridge, "What's so great about 'grep'? Implications for program comprehension tools," 1997.
- [8] W. Norman, B. Michelle, P. Henry, R. Vaclav, and P. LaTrevia, "A comparison of methods for locating features in legacy software," *Journal of Systems and Software*, vol. 65, pp. 105-114, 2003.
- [9] T. Eisenbarth, R. Koschke, and D. Simon, "Aiding program comprehension by static and dynamic feature analysis,"

presented at Proceedings of the IEEE International Conference on Software Maintenance. , 2001.

[10] J. Singer, T. Lethbridge, N. Vinson, and N. Anquetil, "An examination of software engineering work practices " in *Proceedings of the 1997 Conference of the Centre for Advanced Studies on collaborative research* Toronto, Ontario, Canada IBM Press, 1997 pp. 21

[11] R. Prieto-Diaz, "Implementing faceted classification for software reuse " *Communications of the ACM*, vol. 34, pp. 88-97, 1991.

[12] G. Fischer, S. Henninger, and D. Redmiles, "Cognitive tools for locating and comprehending software objects for reuse " in *Proceedings of the 13th International Conference on Software Engineering* Austin, Texas, United States IEEE Computer Society Press, 1991 pp. 318-328

[13] V. M. Sue and L. A. Ritter, *Conducting online surveys*: Sage Publications, 2007.

[14] L. T. Su, "A comprehensive and systematic model of user evaluation of web search engines: I. Theory and background " *Journal of the American Society for Information Science & Technology*, vol. 54 pp. 1175-1192 2003

[15] S. Y. Rieh and N. J. Belkin, "Understanding judgment of information quality and cognitive authority in the www," presented at Proceedings of the ASIS Annual Meeting, Pittsburg, PA, 1998.

[16] D. A. deVaus, *Surveys in social research*, Fourth Edition. ed. London: UCL Press, 1996.

[17] M. B. Miles and A. M. Huberman, *Qualitative data analysis*: Sage Publication, 1994.

[18] A. Strauss and J. Corbin, *Basics of qualitative research: Grounded theory procedures and technique*. Thousand Oaks: Sage Publications, 1990.

[19] J. Pruitt and J. Grudin, "Personas: Practice and theory " presented at Proceedings of the 2003 Conference on Designing for user experiences San Francisco, California 2003

[20] S. E. Crudge and F. C. Johnson, "Using the information seeker to elicit construct models for search engine evaluation " *Journal of the American Society for Information Science & Technology*, vol. 55 pp. 794-806 2004

Appendix A: Text of the Survey

1. How often do you search for source code on the Internet?

- Never
- Occasionally
- Frequently
- *no answer*

2. What information sources do you use when searching for source code on the Internet? (Check all that apply)

- General purpose search engines (such as Google, Yahoo, MSN search)
- Source code specific search engines (such as Koders.com, CodeCrawler)
- Websites - such as sourceforge.net and freshmeat.net
- Your domain knowledge
- References from peers
- Mailing lists
- News websites such as slashdot.org
- Other:

3. Which search sites do you use to search for source code?

- Koders.com
- CodeCrawler

- Sourceforge.net
- Freshmeat.net
- Others:

4. Which of the following programming and/or scripting languages have you had some experience with? (Check all that apply)

- Java
- C/C++
- Perl
- Python
- JSP
- Tcl
- PHP
- Ruby
- Other:

5. Please describe one or two scenarios when you were looking for source code on the Internet.

(Please address details such as: What were you trying to find? How did you formulate your queries? What information sources did you use to find the source code? Which implementation language were you working with? What criteria did you use to decide on the best match?)

6. After searching, once you have identified some possible candidates, what are the criteria that guide you to finally select source code that you will use? (Check all that apply)

- Price
- Terms of license
- Size of source code
- Available functionality
- Level of project activity
- Amount of user support available
- Other:

7. While using the retrieved source code to develop software, which information sources do you consult? Please rank the options by assigning a number to each, from 1 (Most Frequently) to 4 (Least Frequently). Please do not repeat ranks.

- Documentation:
- Other people:
- Mailing lists:
- Chat / Instant messenger:

8. If you could have the ideal search engine while searching for source code on the Internet, what additional features would you like to have?

(Do not worry about implementation details)

9. For how many years have you been developing software professionally?

10. Have you ever contributed source code to an open source project?

- Yes
- No
- *no answer*

11. What is the typical team size of projects that you are involved with? (Both open source as well as proprietary)

- Small (1 - 5 people)
- Medium (5 - 25 people)
- Large (Greater than 25 people)
- *no answer*

12. What is your primary job responsibility?

- Project Management
- Requirements/Product Management
- Software Architecture
- Software Development (i.e. Programming)
- Testing/Quality Assurance
- Technical Writing
- Other:
- *no answer*

13. Where did you hear about this survey?

- Email
- Slashdot.org
- Javaworld.com
- Sourceforge.net
- Freshmeat.net
- Other:

14. Please specify your age: (Optional)
15. Please specify your gender: (Optional)
16. Any suggestions/comments/feedback?