Chapter 3
Archetypal Internet-Scale Source Code Searching

Medha Umarji and Susan Elliott Sim

Abstract To gain a better understanding of what, how, and why programmers search for code on the Internet, 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 major archetypes and one minor archetype, as well as, a range of sizes for search targets. The first archetype was searching for source code that could be excised and dropped into a project. The second archetype was searching for examples of source code to provide information, for example, using the World Wide Web as an enormous desk reference. The targets of these searches could vary in size from a few lines of code to an entire system. The minor archetype was searching for reports and repairs of defects. Factors affecting the final selection of a candidate piece of code included: peer recommendations; availability of help from other programmers; and the level of activity on a project.

3.1 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 in 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, csourcesearch.net with over

M. Umarji (✉)
Symantec Corporation, Mountain View, CA, USA
e-mail: medha_umarji@symantec.com

S.E. Sim
Many Roads Studios, Toronto, ON, Canada
e-mail: ses@drsusansim.org

S.E. Sim and R.E. Gallardo-Valencia (eds.), *Finding Source Code on the Web for Remix and Reuse*, DOI 10.1007/978-1-4614-6596-6_3,

283 MLOC, and Google Code Search with over 1 billion lines of code. 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. But, there are other kinds of searches that can take place on the Internet and we need to know more.

This study was conducted to characterize 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 [3], and Sim, Clarke, and Holt [12]. Using an online questionnaire, we collected data from over 70 programmers who were solicited using Google Groups 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 archetypes and one minor one. The first archetype was searching for a piece of code that can be reused. For example, a text search engine, or a graphical user interface (GUI) widget. The second archetype was searching for reference information, that is, for examples of code to learn from. In this archetype, developers are using the World Wide Web as a very large desk reference. The minor archetype was searching for reports and repairs of bugs, i.e. patches. The two major archetypes had search targets that varied in size, while the minor one did not. The search targets could be small-grained, such as a block of code, medium-grained, such as a package, or large-grained, such as an entire system. The results reported in chapter are an extension of the work reported in an earlier paper [21].

3.2 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 do not 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.

3.2.1 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 10 years or so that it has become commonplace. For-profit corporations are now contributing source code and person-hours to open source projects [5]. The most obvious benefit of the open source movement is that it makes available a "rich base of reusable software" [15].

Communities of practice have evolved from this sharing of programs and knowledge amongst people having common goals and interests, within the open source world. 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 [6]. Artifacts, advice/tips and other relevant knowledge are contributed by members to provide a shared repertoire of resources for the community.

In the open source world, project hosting sites, technology-specific mailing lists and social networking sites are examples of such communities of practice. Sourceforge.net and freshmeat.net host thousands of projects and have an infrastructure that supports the sharing of programs and knowledge. The infrastructure for these projects is provided by developers, and so is the source code – all through extensive collaboration over individual projects.

Technology specific mailing lists such as PHP.net and CPAN.org are a compilation of code snippets, bug reports, patches, discussions and how-to guides related to a specific technology or programming language. These sites are frequented by developers who are interested in learning a particular language or technology, or building on top of it. The lists contain 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. Since it is the culture in open source to share software developed by using open source technology, these archives of source code are increasing exponentially.

Blogs, social bookmarking and other social networking sites have the capability to tag websites containing source code relevant to a particular topic and are excellent sources of reference on latest technologies and trends.

3.2.2 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. However, they are

effective in broad searches for functionality, when good search terms are available. 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 index public source code, cache it in a repository and enable users to search based on a variety of attributes such as class/function names, code licenses, programming languages 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. Like Google Code Search, Koders has options for searching by language and license. 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 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.

To leverage the advantages afforded by open source code, we need search capabilities that are closely integrated with the way that software is developed in open source. The code search engines do not support the social interaction processes that are the lifeline of any project. For example, they do not search for keywords within mailing lists or forums related to a particular topic, users have to use a general-purpose search engine for that purpose. Neither do they support the formation and sustenance of communities of practice that are so essential for learning and sharing in any domain [6].

3.2.3 Source Code Searching

A study of software engineering work practices by Singer et al. [14] found that searching was the most common activity for software engineers. 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 et al. [12] found that the search goals cited frequently by developers were code reuse, defect repair, program understanding, feature addition and impact analysis.

Source code searching for program comprehension involves matching of words or code snippets within an IDE or source code module to a search term, typically using the Unix-based grep facility, the find command in Unix and also the File Find command under Microsoft Windows [12]. It was also found that programmers used only strings or regular expressions to form their search terms, even though they were searching for semantically significant pieces of code. Grep is by far the most popular search facility due to ease of specification of search terms, a command-line interface, and a direct match with the search model of the programmer [13]. Programmers trust grep because it is successful most of the times, and the cost of failure is insignificant.

Program comprehension tools can be categorized as (i) extraction tools such as parsers (Rigi) [8], (ii) analysis tools for clustering, feature identification and slicing (Bauhaus tool [2]), and (iii) as presentation tools such as code editors, browsers and visualizations [16, 20].

Lethbridge et al. [13] in their study on grep discuss that searching within source code is used for locating the bug/problem, finding ways to fix it and then evaluating the impact on other segments. Sim et al. [12] found that programmers were most frequently looking for function definitions, variable definitions, all uses of a function and all uses of a variable.

However, none of these existing tools have capabilities to search for software components based on functionality and purpose – which is the basic idea behind Internet-scale source code searching.

3.2.4 Software Reuse

It is evident from the discussion so far that 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.

Reuse is a common motivation for Internet-scale source code searching [15]. Programmers do not want to "re-invent the wheel," especially when the open source world allows reuse to occur at all levels of granularity, starting from a few lines of code, to an entire library; from a tool to an entire system.

Reuse in proprietary settings involved indexing and storing software components in a way that would make retrieval and usage easy (for example, the structured classification technique, by Prieto-Diaz [10]). Complex queries had to be formed to retrieve such components and the process of translating requirements into search terms posed a cognitive burden for software engineers. Fischer et al. [4] also discuss 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. They also discuss the technique of retrieval by reformulation – a continuous refinement process that forms cues for retrieval of components that are not well-defined initially.

The problem of discourse persists through the open source era as the primary method of searching continues to be keywords and regular expressions. Support provided for locating and comprehending software objects does not scale up to the actual potential for reuse even in open source projects.

Reuse of open source code occurs with an understanding that effort will be expended in contextualizing, comprehending and modifying a piece of software – while traditionally, the reuse concept assumed little or no modification of components. Another interesting difference is that in open source the options available for a given search query are tremendous as opposed to a company-wide repository of source code, which may or may not have relevant reusable code.

3.3 Method

Online surveys have become increasingly common over the last decade, as Internet usage has grown by leaps and bounds. Surveys have become an established empirical method, especially for human behavior on the Internet [19].

These 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 [9, 11, 18, 22]. Sim, Clarke, and Holt [12] 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 [1]. The design of this study is presented in this section.

3.3.1 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.

3.3.2 Data Collection

We designed an online survey with 11 closed-ended questions and 2 open-ended questions. This chapter is focused on the results from one of the open-ended questions, which asked:

> 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?)

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.

The survey was open for 6 months in 2006–2007 to collect responses. Invitations to participate in the survey were posted to the Javaworld mailing list, and the following mailing lists beginners-cgi@perl.org, comp.software-engg, comp.lang.c, and comp.lang.java. We chose these web sites, because had they had users with a variety of interests, the discussions were high technical in nature, and there was little overlap between the groups.

3.3.3 Data Analysis

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 [7] and a grounded theory approach [17]. 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.

3.3.4 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 a 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.

3.4 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 (77 %), C++ (83 %) and Perl (60 %). A few had contributed to an open source project, though most were users of open source Applications. Within the criteria guiding final selection of source code, 77 % users based their decisions on available functionality, 43 % considered the licensing terms and 30 % considered the amount of user support available. Amongst the information sources consulted while searching for source code, documentation ranked highest, followed by mailing lists and other people. Most of the respondents (59 %) had experience working on small teams with 1–5 people.

3.4.1 Situations

We analyzed 58 scenarios of source code searching. They ranged in length from one to ten lines. Figure 3.1 below is an example of a good response that we received.

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.

> 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.

Fig. 3.1 Example search description

3.4.2 Object of the Search

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 (Table 3.1). 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 Fig. 3.1, the motivation was coded as "reference example" and the size of search target was coded as "subsystem".

Code for reuse	As-is reuse	Reference example	Row total
Block	8	4	12
Subsystem	21	11	32
System	5	2	7
Column total	37	22	51

Table 3.1: Purpose by target size

3.4.2.1 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 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 that was 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," and only used the other implementations for ideas.

3.4.2.2 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 a 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 on 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."

3.4.3 Process of Search

With respect to the process of search, anecdotes were categorized on the starting point for the search, the tools used, and the criteria used to make the final selection. In Fig. 3.1, the starting point was a recommendation from a friend, the tool used was search.cpan.org, and no selection criterion was mentioned.

3.4.3.1 Starting Point for Search

A common starting point for Internet-scale code searches was recommendations from friends to use a particular piece of software. Other potential starting points were reviews, articles, blogs, and social tagging sites. When no such starting point was available, programmers went straight to search tools.

3.4.3.2 Search Tools Used

By far, the most popular tool for finding source code on the web was general-purpose search engines, such as Google and Yahoo! The search feature on specific web sites and archives was also popular. Interestingly, the source code-specific search engines were used only occasionally.

3.4.3.3 Selection Criteria

A number of common themes also emerged among the criteria that programmers used to make a final selection among different options. Often the choices were limited, so there were few degrees of freedom in the final selection. Criteria that were mentioned by the respondents were level of activity on the project, availability of working experience, availability of documentation, and ease of installation. Surprisingly, code quality and the user license for the source code were low priorities in the selection criteria.

The results are presented as archetypal searches in Sect. 3.5 and as observations about the search process in Sect. 3.6.

3.5 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.

3.5.1 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 reimplementation, 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 tested 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 can be in the form of of a report or post to a form 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.

3.5.2 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.

3.6 Discussion

In this section, we explore how social interaction processes supported by the right search tools can help programmers to arrive at the right code snippet, component or exemplar.

As discussed previously, a typical search begins with a cue such as advice from a colleague or use of the immensely popular general purpose search engines.

Search mechanism	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 3.2: Tools/information sources used in search

Our survey showed that 60 of the 69 respondents used general purpose search engines (refer to Table 3.2). More than half the respondents relied on their domain knowledge to find the right source code. Project hosting sites came next, with 34 of the respondents using them for source code search.

Elements from the social network were used frequently especially peer references (30) and mailing lists (16). In our descriptive data we observed that social tagging sites (del.icio.us) and compilation sites created by a group of programmers featured often.

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

3.6.1 Documentation

An initial assessment is done using web pages and documentation, as one user put it, "... (the core developer) had a 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 it can't be easily determined whether it has the required features and documentation. The basic functionality has to be in place, and supported by requisite documentation.

3.6.2 Peer Recommendations

Peer recommendations were the most trusted and valued – especially if the person has used the software before. For instance, one respondent stated that "... friends recommended Graph.pm; searched for that on search.cpan.org, and found it was just what I needed." In the absence or inaccessibility of peer advice programmers then look for availability of help from people within their online social network, or within the project context.

3.6.3 Help from One's Social Network

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.

Availability of help is also determined from the project activity. Respondents preferred large open source projects that were very active. For example, "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.

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 code quality (i.e. whether the code is peer reviewed and tested), and reliability (Table 3.3).

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

Table 3.3: Criteria for selecting a code component

3.6.4 Feature Suggestions

Developers not only look for code, they also look for a social system through which they can contribute their knowledge and expertise as well as learn from their counterparts. As reported in Table 3.2, the domain knowledge and social networking are key ingredients of the search process in addition to search engines.

We also observed that programmers use social tagging websites for technical information and applications. The current source code repositories should be appended with a recommender system wherein programmers could obtain not just code components, but also real subjective opinions of people who have used those components.

3.7 Conclusions

The goal of this research study was to gain an understanding of Internet-scale source code searching in order to inform the design and evaluation of tools for web-based source code retrieval. We observed that programmers mainly search for either reusable components or reference examples. The granularity of search targets varies from a block of code to an entire system. Some directions for future research in this area are: Which search engines are better than others with respect to code search? Does search engine performance depend on types of tasks?

References

[1] D.A. deVaus. *Surveys in Social Research*. UCL Press, London, fourth edition. edition, 1996.

[2] T. Eisenbarth, R. Koschke, and D. Simon. Aiding program comprehension by static and dynamic feature analysis. In *Proceedings of the IEEE International Conference on Software Maintenance.*, pages 602–611, 2001. TY - CONF.

[3] Mark Eisenstadt. My hairiest bug war stories. *Communications of the ACM*, 40(4):30–37, 1997.

[4] Gerhard Fischer, Scott Henninger, and David Redmiles. Cognitive tools for locating and comprehending software objects for reuse. In *Proceedings of the 13th international conference on Software engineering*, pages 318–328, Austin, Texas, United States, 1991. IEEE Computer Society Press.

[5] R. Goldman and R.P. Gabriel. *Innovation Happens Elsewhere: Open Source as Business Strategy*. Morgan Kaufmann Publishers, San Francisco, CA, 2005.

[6] Jean Lave and Etienne Wenger. *Situated Learning: legitimate peripheral participation*. Cambridge University Press, England, 1991.

[7] M. B. Miles and A. M. Huberman. *Qualitative data analysis*. Sage Publications, Thousand Oaks, CA, 1994.

[8] H. A. Muller and K. Klashinsky. Rigi-a system for programming-in-the-large. In *Proceedings of the 10th international conference on Software engineering*, pages 80–86, Singapore, 1988. IEEE Computer Society Press.

[9] E. B. Parker and W. J. Paisley. Information retrieval as a receiver-controlled communication system. *Education for Information Science*, pages 23–31, 1965.

[10] Ruben Prieto-Diaz. Implementing faceted classification for software reuse. *Communications of the ACM*, 34(5):88–97, 1991.

[11] Soo Young Rieh. Judgment of information quality and cognitive authority in the web. *Journal of the American Society for Information Science & Technology*, 53(2):145–161, 2002.

[12] 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*, page 180, Los Alamitos, CA, 1998. IEEE Computer Society.

[13] Janice Singer and Timothy Lethbridge. What's so great about 'grep'? implications for program comprehension tools. Technical report, National Research Council, Canada, 1997.

[14] Janice Singer, Timothy Lethbridge, Norman Vinson, and Nicolas Anquetil. An examination of software engineering work practices. In *Proceedings of the 1997 conference of the Centre for Advanced Studies on Collaborative research*, page 21, Toronto, Ontario, Canada, 1997. IBM Press.

[15] Diomidis Spinellis and Clemens Szyperski. Guest editors' introduction: How is open source affecting software development? *IEEE Software*, 21(1):28–33, 2004.

[16] Margaret-Anne D. Storey. Theories, tools and research methods in program comprehension: past, present and future. *Software Quality Journal*, 14(3): 187–208, 2006.

[17] Anselm Strauss and Juliet Corbin. *Basics of Qualitative Research: Grounded Theory Procedures and Technique*. Sage Publications, Thousand Oaks, 1990.

[18] Louise 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*, 54(13):1175–1192, 2003.

[19] V. M. Sue and L. A. Ritter. *Conducting Online Surveys*. Sage Publications, Thousand Oaks, CA, 2007.

[20] Scott R. Tilley, Dennis B. Smith, and Santanu Paul. Towards a framework for program understanding. In *Proceedings of the 4th International Workshop on Program Comprehension (WPC '96)*, page 19. IEEE Computer Society, 1996.

[21] Medha Umarji, Susan Elliott Sim, and Cristina V. Lopes. Archetypal internet-scale source code searching. In Barbara Russo, editor, *OSS*, page 7, New York, NY, 2008. Springer.

[22] H. Wang, M. Xie, and T. N. Goh. Service quality of internet search engines. *Journal of Information Science*, 25(6):499–507, 1999.