Software search is not a science, even among scientists: A survey of how scientists and engineers find software

M. Hucka[a,*], M. J. Graham[b]

[a] *Department of Computing and Mathematical Sciences, Mail Code 306-15, California Institute of Technology, Pasadena, California 91125, USA*
[b] *Department of Astronomy, Mail Code 158-79, California Institute of Technology, Pasadena, California 91125, USA*

Abstract

Improved software discovery is a prerequisite for greater software reuse: after all, if someone cannot find software for a particular task, they cannot reuse it. Understanding people's approaches and preferences when they look for software could help improve facilities for software discovery, but there is a surprising shortage of data about what people do in practice. We surveyed users in several technical fields to better understand their approaches and selection criteria. We found that even among highly trained people, the relatively unsophisticated approaches of relying on general Web searches, the opinions of colleagues, and the literature were the most-often used approaches. However, developers were more likely than non-developers to search in community sites such as Stack Overflow and GitHub, even when seeking ready-to-run software rather than source code. Our results also reveal characteristics that matter to people searching for software, as well as factors that can prevent people from finding software.

Keywords: software search, software reuse, software catalogues, survey

1. Introduction

Software is critical to research (Hannay et al., 2009; Hettrick, 2014; Howison and Bullard, 2015; Howison et al., 2015; Ince et al., 2012; Katz et al.,

*Corresponding author.
Email addresses: mhucka@caltech.edu (M. Hucka), mjg@caltech.edu (M. J. Graham)

2016; Katz and Ramnath, 2015; Morin et al., 2012; Stewart et al., 2013; Wilson, 2006), yet finding software suitable for a given purpose remains surprisingly difficult (Bourne, 2015; Cannata et al., 2005; Howison and Bullard, 2015; White et al., 2014). Few resources exist to help users discover available options or understand the differences between them (White et al., 2014). A recent study (Bauer et al., 2014) of developers at the Internet search company Google underscored the depth of the problem: the authors found the factor "most disruptive to the [software] reuse process" was "difficulties in finding artifacts." In other words, *even developers at Google have difficulty finding software.*

Searching the Internet with a general-purpose search engine has previously been reported to be one of the most popular approaches (Samadi et al., 2004; Umarji et al., 2008). Despite its popularity, this approach suffers from demonstrable problems. It requires devising appropriate search terms, which can be challenging for someone not already familiar with a given topic or who is not a native English speaker. Web searches also can yield dozens of viable candidates mixed with thousands of irrelevant results, requiring the user to follow links and examine individual candidates—a time-consuming and tedious task. Finally, some questions cannot be answered through Web searches without substantial additional effort, such as what are the differences between candidate software tools. Other approaches to finding software, such as looking in the literature or asking on social media, suffer from still other problems such as the potential for incomplete or biased answers. The difficulty of finding software and the lack of better resources brings the potential for duplication of work, reduced scientific reproducibility, and poor return on investment by funding agencies (Cannata et al., 2005; Council, 2003; Crook et al., 2013; Niemeyer et al., 2016; Poisot, 2015; White et al., 2014).

One of the first steps to providing more effective resources for finding software is to understand factors that influence how users locate and select software today. However, most prior work on this topic has focused on *developers* searching for *source code*; few studies included non-developers or asked how people look for ready-to-run software (rather than source code). In addition, prior work has

examined the use of *search* to find software, but not other options such as the use of catalogs or indexes. In an effort to understand these and other aspects of how people find software, in late 2015 we conducted a survey involving members of numerous mailing lists in astronomy and systems biology. In this article, we report on five of the research questions addressed by our survey:

RQ1 How do people look for ready-to-run software?

RQ2 What criteria do people use when choosing ready-to-run software?

RQ3 What information would people like to find in a catalog of software?

RQ4 How do software developers look for source code?

RQ5 What can prevent developers from finding suitable source code?

This survey contributes to the body of research into the reuse of software, particularly how scientific users locate software. It provides insights into the current practices and experiences in searching for software from two distinct groups of people: those looking for ready-to-run software and those looking for software source code. These results can inform the future development of improved resources to help users, especially scientific users, discover software.

The remainder of this article is divided as follows. In Section 2, we overview related work. In Section 3, we describe our survey design and research methods, while in Section 4, we report our results. We discussion the results, implications and limitations in Section 5, and conclude with Section 6.

2. Related work

Some of the our research questions have been addressed in other studies in the literature, though not all questions have been previously addressed or examined in the same context. We discuss relevant work in this section.

2.1. Surveys examining how people find ready-to-run software

Many surveys have examined software developers and search characteristics in the context of software *code* reuse, but few have examined how users—whether they are developers or not—go about locating and selecting *ready-to-run* software. Our research uncovered only three reports of surveys that were not focused specifically on a software development context.

Joppa et al. (2013) surveyed 596 scientists working on modeling biological species distribution, and asked them what software they used and why they chose it. The reasons given by the respondents provide some insight into how the scientists found the software they used. In order of popularity, the answers that mentioned something about "how" were: "I tried lots of software and this is the best" (18% of respondents), "Recommendation from close colleagues" (18%), "Personal recommendation" (9%), "Other" (9%), "Recommendation through a training course" (7%), "Because of a good presentation and/or paper I saw" (4%), and "A reviewer suggested I use it" (1%). Surprisingly, none of the responses in Joppa et al. survey explicitly mentioned searching the Internet, although it is possible that some of the answers such as "I tried lots of software and this is the best" and "Other" subsumed the use of Web searches.

Huang et al. (2013) summarized interviews of 15 students and faculty in bioinformatics. They found that four factors influenced the software selection: (1) suggestions from senior members of an institution; (2) mentor involvement in the tool's development; (3) the number of publications *about* the software; and (4) the software's reputation, based on the number of publications mentioning the *use* of the tool. Unfortunately, Huang et al.'s report does not include any quantitative or qualitative data about the relative importance of these factors.

Lawrence et al. (2014, 2015) conducted a large survey about the use of science gateways by members of scientific communities. Several of their questions and results are relevant to the topics of our own survey:

- They asked participants to indicate domains of expertise. The top five were "Physical and Mathematical Sciences" (30%), "Life Sciences" (22%),

"Computer and Information Sciences" (16%), "Engineering" (16%), and "Environmental Sciences" (14%), though 16% did not indicate a domain.

- Lawrence et al. asked how people learn about and choose science gateways—a question related to our RQ1. They found that 78% indicated they learned about technologies from colleagues, 61% indicated conferences and other meetings as a source, 51% said publications, 38% said Web searches and speciality sites, 33% from students, and less than 10% from mailing lists or other methods such as magazine advertisements.

- Related to our RQ4, they asked software developers "How do you keep up to date with web-based technologies?", limiting answers to two choices from a predefined list and a free-text "Other" field. The three most popular answers were: using online communities via email or Web-based forums (47%), one's own development team (43%), and focused workshops (18%).

- In another question, Lawrence et al. (2015) asked participants "Assuming cost is not a factor, what are the most important factors you consider when adopting a new technology? Please select the three (3) most important factors in your decision-making process". Since this question had direct relevance to RQ2 in our survey, we include the full response results here:

  - "Documentation available" (49%)
  - "Ability to Adapt/Customize" (35%)
  - "Demonstrated Production-Quality Reliability" (31%)
  - "Availability of Technical Support" (30%)
  - "Open Source" (27%)
  - "Existing User Community" (20%)
  - "Interoperability with Other Systems" (20%)
  - "Availability of Support for Bug Fixes & Requests" (19%)
  - "Testimonials/User Ratings" (16%)
  - "Project Longevity" (13%)

– "Licensing Requirements" (12%)
– "Availability of Long-Term Maintenance" (11%)
– "Reputation of Those Who Built the Software" (11%)

2.2. Surveys examining the use of catalogs of software

A potential aid to finding software is a catalog or index that includes known software and allows people to browse and search by various criteria (Allen and Schmidt, 2015; Katz, 2015; Marshall et al., 2006; Mena et al., 2006; White et al., 2014). Numerous public software catalogs exist; most are domain/community-specific (e.g., Allen et al., 2012; Bönisch et al., 2013; Browne et al., 1995; Gleeson, 2016; Hempel et al., 2016; Hucka et al., 2016; National Aeronautics and Space Administration, 2016; Noy et al., 2009; Shen, 2015), though some general indexes also exist (e.g., Black Duck Software, Inc., 2016; Johansson and Olausson, 2016; Mario, 2016; SlashDot Media, 2016). Surveys that examined people's approaches to finding software sometimes posed questions about the use of catalogs, but these invariably concerned *whether* or *how often* users employed catalogs, not what information they sought in the catalog. For example, the survey by Lawrence et al. (2015) touches on the topic, since gateways often provide some kind of listing of accessible software; however, the authors do not report on how users employed the lists that may have been available. Marshall et al. (2006) considered the question of whether users employed catalogs, but did not report the characteristics of those catalogs such as the information they contained. We could find no directly-related work to compare to our RQ3.

2.3. Surveys examining how developers find source code

Most studies of how users find software have done so in the context of software development and the reuse of software source code. The types of reuse in these situations range from black-box reuse of libraries or other software components (i.e., reusing code "as-is"), to reuse of code fragments; in addition, in programming contexts, many studies examined the reuse of other kinds of artifacts such as documentation, specifications, architectural patterns, and more.

Samadi et al. (2004) reported preliminary findings from a survey conducted by the NASA Earth Science Software Reuse Working Group. Their survey was distributed to government employees and contractors in the Earth science community. Some results from the study are pertinent to our research question RQ4. First, on the topic of how people found reusable software artifacts, the following approaches were noted: (1) word of mouth or personal experiences from past projects, (2) general Web search engines (e.g., Google), and (3) catalogs and repositories. The authors report "Generic search tools (such as Google) were rated as somewhat important, whereas specialist reuse catalogs or repositories were not cited as being particularly important". Second, for criteria used to decide which specific components to choose, the authors report that "most respondents chose saving time/money and ensuring reliability as their primary drivers for reuse". The following additional considerations were noted: (1) "ease of adaption/integration", (2) availability of source code", (3) "cost of creating/acquiring alternative", and (4) "recommendation from a colleague". The authors found that (a) availability of support, (b) standards compliance, and (c) testing/certification, were "not ranked as particularly important".

Samadi et al.'s study was reprised in 2005 with a wider audience that included members of academia (Marshall et al., 2006). The authors reported that the survey produced essentially similar results to their 2004 survey. Marshall et al. noted that the primary reason given by people for not reusing software from outside of their group was "they did not know where to look for reusable artifacts and they did not know suitable artifacts existed at the time." For those who *did* engage in reuse, "personal knowledge from past projects and word-of-mouth or networking were the primary ways of locating and acquiring software development artifacts." On the topic of how people located software, Marshall et al. noted "the use of reuse catalogs and repositories was rated the most important method of increasing the level of reuse within the community."

In a different NASA study, Orrego and Mundy (Orrego and Mundy, 2007) examined software reuse in the context of flight control systems. They studied 63 projects using interviews, surveys and case studies. In interviews with 15

people, the difficulty of assessing the characteristics of software was stated as the most problematic aspect of reusing software, usually because of inadequate documentation for the software to be reused.

Umarji et al. (2008) (also discussed by Umarji and Sim, 2013) surveyed Java programmers in 2006–2007 to understand how and why they searched for source code. Using invitations to mailing lists and newsgroups, they solicited participation to fill out a Web survey, and received 69 responses. Several facets of the Umarji et al. study are especially relevant to our own survey:

- The found common starting points for searches to be (1) recommendations from friends, and (2) reviews, articles, blogs and social tagging sites.

- With respect to how developers conducted searches, the participants in the survey used the following, in order of popularity: (1) general-purpose search engines (87% of participants), (2) personal domain knowledge (54%), (3) project hosting sites such as SourceForge.net (SlashDot Media, 1999) (49%), (4) references from peers (43%), (4) mailing lists (23%), and (5) code-specific search engines such as Google Code Search (Google, Inc., 2006) (16%).

- With respect to the selection criteria used by developers to choose a solution, Umarji and Sim (2013) report that the most important factors were: (1) software functionality (78%), (2) type of software license (43%), (3) price (38%), (4) amount of user support available (30%), and (5) level of project activity (26%).

Singer et al. (2014) examined how software developers active on GitHub use Twitter. They conducted an initial exploratory survey with 271 GitHub users (270 of whom said they develop software) and followed it up with a validation survey involving 1,413 GitHub users (1,412 of whom said they develop software). Their results have the following relevance to the topic of how people find and choose software. Developers extend their knowledge of software (including new software tools and components) by asking and answering questions, partic-

ipating in conversations, and following experts. This can lead to serendipitous discovery of reusable methods, software components and software tools. Singer et al. noted "Some developers mentioned that Twitter helps them find and learn about things that they would not have been able to search for themselves, such as emerging technologies that are too new to appear in web searches."

Bauer et al. (2014) studied reuse practices by developers at Google. Several questions in Bauer et al.'s survey are relevant to the present study:

- They asked subjects for their top three ways of sharing software components. They received 63 responses: common repository (97%), packaged libraries (34%), tutorials (31%), blogs (19%), email (9%), "I do not share artifacts" (3%), and "other" (3%).

- Bauer et al. asked about the preferred ways to find reusable software. They received 106 responses: code search (77%), communication with colleagues (64%), Web search (49%), browsing repositories (41%), browsing documentation (23%), "other" (8%), "code completion" (5%), code recommendation systems (3%), and tutorials (3%).

- They also asked "What do you do to properly understand and adequately select reusable artifacts?" and received 115 responses: interface documentation (72%), examples of usage on blogs and tutorials (64%), reviewing implementations (64%), reading guidelines (51%), exploring third-party products (28%), "other" (10%), and participating in training for third-party products (5%).

2.4. Other studies

A number of other studies have examined code search by developers (e.g., Gallardo-Valencia and Sim, 2011; Sadowski et al., 2015; Sim et al., 2012; Singer et al., 1997; Xia et al., 2017) or other questions surrounding code reuse (e.g., Frakes and Fox, 1995; Morisio et al., 2002; Sojer and Henkel, 2010). However, these other studies do not have results that we can compare to our specific research questions RQ1–RQ5. For example, Sadowski et al. (2015) report on how

developers search for code, but their research questions focused on such things as the properties of search queries. These and other studies provide important but complementary data to our results, and in the interest of conciseness, we do not report on them further here.

Finally, there are other works here that have applied other methods (i.e., not surveys) to examining developer behavior. The two main classes of non-survey techniques have been the analysis of search engine logs (Bajracharya and Lopes, 2009, 2012; Brandt et al., 2010, 2009; Ge et al., 2014; Jansen and Spink, 2006; Li et al., 2009; Teevan et al., 2004; Völske et al., 2015), and observational studies (often coupled with interviews), either in institutional settings or in laboratory environments (Banker et al., 1993; Brandt et al., 2009; Dabbish et al., 2012; Gallardo-Valencia and Sim, 2013; Murphy-Hill et al., 2015; Pohthong and Budgen, 2001; Sherif and Vinze, 2003; Sim et al., 2013; Sim and Alspaugh, 2011; Sim et al., 2011). Due to the different techniques and intentions behind these efforts, the results are outside the scope of what we can compare meaningfully to the present survey.

3. Survey design and methods

Our survey was designed to shed light on current practices and experiences in searching for software in two different situations: looking for ready-to-run software, and looking for software source code. Respondents did not have to be software developers themselves (although the results show that most were). We chose to use a Web-based survey because it is an approach that (1) is well-suited to gathering information quickly from a wide audience, (2) requires modest development effort, and (3) can produce data that can be analyzed qualitatively and quantitatively.

3.1. Instrument development

Following the practices of other surveys in computing (e.g., Kitchenham and Pfleeger, 2008; Varnell-Sarjeant et al., 2015), we designed the instrument iteratively and paid attention to the following points:

- Wording. We sought to make the questions clear and unambiguous, and avoid implying a particular perspective. We elaborated each question with explanatory text under the question itself.
- Relevance to user's experiences. We limited our questions to topics that could reasonably be assumed to be within the experiences of our audience.
- Contemporary circumstances. We tried to ground the questions by referring to real resources and specific software characteristics that we believe are relevant to computer users today.
- Ethics. We avoided questions that might be construed as being too personal or about proprietary policies at participants' place of work.

We performed a pilot survey with a dozen colleagues as subjects. Based on their feedback, we removed or expanded questions as necessary to achieve the final version. The final survey form is presented as a supplementary file. The instrument contained a total of 22 questions (of which 18 were content questions), and included conditional branches so that the final number of questions actually seen by any given respondent depended on the answers selected to certain screening questions. There were five main groups of questions in the survey:

1. Basic demographic and general information, suitable for all respondents.
2. Questions for software users who have the freedom to choose software. This section was only shown if participants indicated that they have some choice in the software they use.
3. Questions for software developers. This section was only shown if respondents indicated that are engaged in software development.
4. Questions for software developers who search for source code. This was only shown if respondents indicated both that they are software developers and that they search for software source code.
5. Survey feedback. This section sought feedback about the survey itself.

Questions in section No. 2 of the survey aimed to establish the relative importance of different search criteria. Those in section Nos. 3 and 4 sought to characterize the experiences of the developer.

The survey form used a mixture of four types of questions: check boxes, pull-down selection menus, two-dimensional rating grids, and short-answer input fields. Some of the questions allowed answers on a nominal scale (for example, approaches used for finding software), some questions used an ordinal scale (for example, the importance of different considerations when looking for software), and some were open-ended questions asking for free-form text.

3.2. Administration

We used Google Forms (Google, Inc., 2015a) to implement the survey instrument. The version of Google Forms was the free edition made available as of September, 2015. We obtained prior approval for the survey protocol from the California Institute of Technology's Committee for the Protection of Human Subjects. The survey form itself included a Web link to a copy of the informed consent form for survey participation. The first question in the survey provided a clickable checkbox by which subjects had to indicate they had read the informed consent form and consented to our use of their responses to the survey. This was the only question in the survey that required a response; all other responses were optional.

We generated a URL (Uniform Resource Locator) for the survey form using Google Shortener (Google, Inc., 2015b), a service that produces shortened URLs and simultaneously provides an analytics facility tied to the URL. On September 1, 2015, we invited participation in the survey. As mentioned below, we advertised the survey on mailing lists and social media oriented to the astronomical and biological sciences, particularly to computational subcommunities within those domains. Recipients were free to participate if they chose. The introduction and instructions for the survey were brief. The survey had no express closing date.

3.3. Sampling plan

We used nonprobabilistic convenience sampling with self-selection. We advertised the survey on electronic mailing lists oriented to the astronomical and biological sciences: the *IVOA* mailing list (astronomy), a Facebook astronomy list, the mailing list `sysbio@caltech.edu` (systems biology), the forum `sbml-interoperability@googlegroups.com` (systems biology), the list `cds-all@caltech.edu` (departmental list), and our immediate work colleagues (totalling a dozen people). Taken together, the members are a mix of staff, students, and faculty working in academia, government laboratories, and industry.

Potential biasing factors in the results include those that are common to self-selected written surveys with convenience sampling: response bias (i.e., people who responded may have different characteristics than those who did not), coverage errors (i.e., the representation of participants may not be balanced across different subcommunities), and item bias (i.e., some questions may have been skipped intentionally or unintentionally). An additional possible source of bias is that the authors are relatively well-known within the subcommunities to which the survey was advertised, which may have influenced respondents.

3.4. Population sample

We analyzed the results obtained by December 31, 2015. We estimate the number of potential recipients of the mail announcements to be at least 2300. The number of completed survey forms was 69. As mentioned above, our survey URL was backed by an analytics facility; this provided the number of URL clicks, the referrer sources, and source geographical locations. According to this facility, the survey form was accessed 172 times. Using these three numbers, we can calculate the following:

1. Estimated access rate to survey form: approximately 7.5% (172/2300).
2. Estimated response rate: approximately 3% (69/2300).

Unfortunately, we cannot be certain of the actual number of recipients. While we can determine the number of addresses we contacted, some of the

addresses on mailing lists may be obsolete or unreachable, or the recipients' electronic mail systems may have filtered out the mail messages. Thus, we can only estimate the response rate.

3.5. Analysis

Descriptive statistics were performed using custom programs written in the language Python (Perez et al., 2011; van Rossum and de Boer, 1991), version 3.4, in combination with the NumPy (Van Der Walt et al., 2011) package, version 1.10.4. The figures in this paper were generated with the help of the Python library Matplotlib (Hunter, 2007), version 1.5.1.

4. Results

4.1. Demographics

Our survey included several questions to gather general demographic information. One of the first questions was "What is your primary field of work?", with multiple choices and "Other" as the answer options. Figure 1 shows the answer choices and the number of responses. Of 69 respondents, 57% identified themselves as working in the physical sciences, 46% in computing and maths, 28% in biological sciences and 7% in a range of others. Subjects could select more than one field, and participants made use of this feature: 17 respondents selected two fields of work, six selected three fields, and one indicated four fields of work.

To assess how computer-intensive people's work activities are, the survey included the question "In your work, on a typical day, approximately what fraction of your time involves using or interacting directly with software on a computer or other computing device?" The answer options were in the form of a pull-down menu with values ranging from 0% (none) to 100% (all), in 5% increments. Note the question was not limited to time spent using technical software—respondents were free to interpret this broadly to mean any software used in a work context. Figure 2 provides a bar graph of the responses. The results show (Figure 2) that the overwhelming majority of our respondents spend

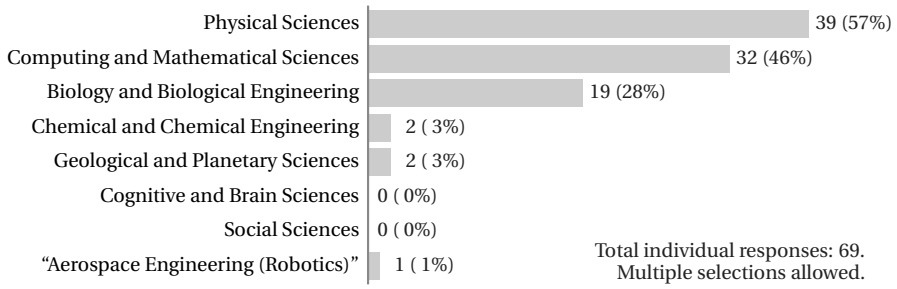

Figure 1: Respondents by discipline. This question offered the first eight predefined categories and a slot for write-in, free-text answers. Choices were nonexclusive. Some respondents wrote write-in answers that were clearly subsumed by one of the predefined categories; in those cases, we adjusted the totals appropriately. One response, "Aerospace Engineering (Robotics)", did not fit any predefined category; we included it as a true "Other" value.

over 50% of their day interacting with software. To quantify this further, assuming a typical 8 hour working day, we can conclude that 94% of participants regularly spent more than four hours of their day engaged with software, and 68% more than six hours.

As mentioned above, the overall motivation for the survey was to understand how people find software. Thus, an important precondition was whether subjects actually had a choice in the software they used. (The rationale for this is that if a person has no choice but to use software that is already provided or selected for them, then their answers to questions about how they find software

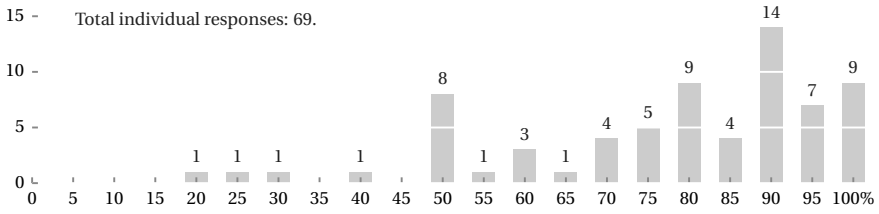

Figure 2: Bar graph of responses to the question "In your work, on a typical day, approximately what fraction of your time involves using or interacting directly with software on a computer or other computing device?"

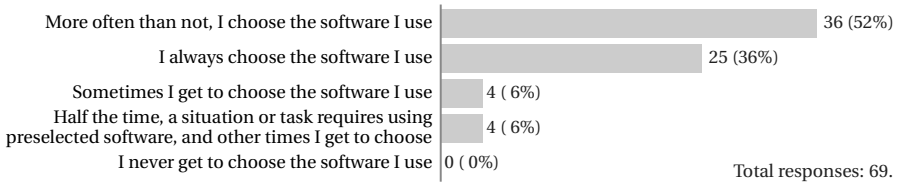

Figure 3: Responses to "In your work, how much freedom do you usually have to choose the software you use?".

would not be meaningful.) This motivated another question in the survey: "In your work, how much freedom do you usually have to choose the software you use?" Answers to this question were used to select subsequent survey questions: if a respondent answered "Never" to this question, then the remaining questions were skipped and people were shown the final survey page. Figure 3 provides the results for this question. The results show that every one of our respondents had some choice in the software they used.

In response to another question, "Are you involved in software development?", 56 (81%) answered "Yes" and 13 (19%) answered "No". The answer to this question controlled the display of additional questions relevant to developers. Among the questions that were made available to the 56 who answered "Yes" were additional demographic questions. (Those who answered "No" were not shown the additional questions or the questions relevant to developers, and were instead taken to the final survey feedback page.) The first question for developers was "For how many years have you been developing software?" with a free-form text field for answers. We manually processed the 56 text responses to remove extraneous text and reduce them to numbers, and then tabulated the values. Figure 4 provides a histogram of the responses received for those who answered the question with an interpretable answer (55 out of 56).

Another question asked of those who indicated they were involved in software development was "In your current (or most recent) software development project, what is (or was) your primary responsibility?" It offered eight multiple

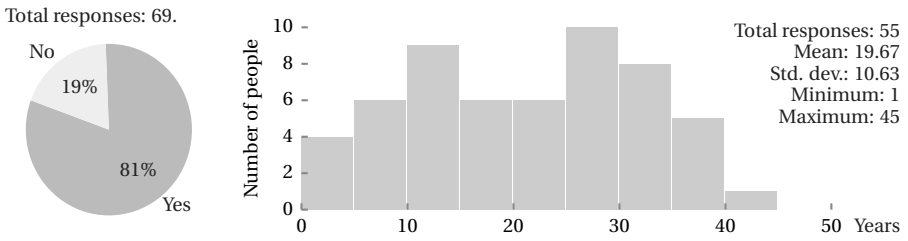

Figure 4: (Left) Responses to the question "Are you involved in software development?" (Right) Histogram plot of years that respondents have been developing software (for those who also answered "Yes" to the question of whether they were involved in software development).

choice items and a ninth "Other" choice with a free-form text field. The choices were nonexclusive: although we asked for people's primary responsibility, participants were free to choose more than one, and the explanatory text for the question indicated "If it is hard to identify a single one, you can indicate more than one below." Figure 5 provides a tally of the responses.

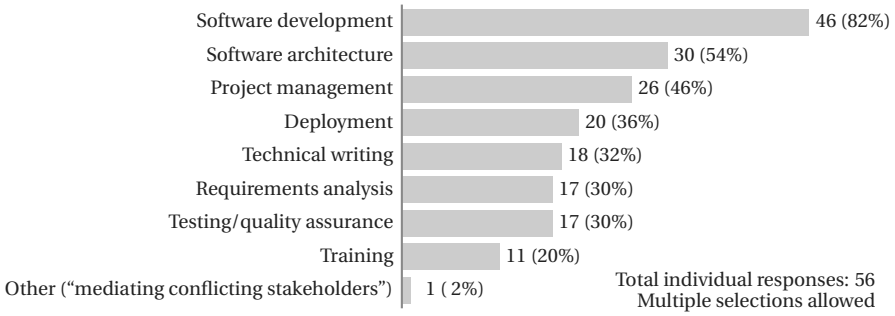

Figure 5: Responses to the question "In your current (or most recent) software development project, what is (or was) your primary responsibility?" This question was shown only to the 56 respondents who answered "Yes" to the question of whether they were involved in software development. This survey question offered the first eight predefined categories and an additional slot for free text under "Other"; only one respondent provide a value for "Other". Choices were nonexclusive.

The survey also posed the question, "What is the typical team size of projects you are involved with?" The form of the answers was again a set of multiple

choice check boxes with an "Other" choice that offered a free-form text field. Answers were provided by all 56 respondents who answered "Yes" to the question of whether they were involved in software development (Figure 4), and none of the participants selected "Other". A total of 43 respondents (77%) selected "Small (1–5 people)", 12 respondents (21%) chose "Medium (6–25 people)", and 1 respondent selected "Large (more than 25 people)".

We also asked "Which programming and/or scripting language(s) have you had the most experience with?" We provided 22 predefined choices along with a free-text "Other" option. Choices were nonexclusive, and the elaboration under the question explicitly requested "Please select up to 3 languages which you have used the most." The top five responses were: Python (selected by 59% of participants), C (50%), Java (34%), shell scripting (32%), and C++ (27%).

Finally, our survey was designed to show a subset of questions of relevance to those developers who indicated they searched for source code. The question "How often do you search online for software source code?" had six answer choices. If respondents chose any option *other* than "Never", they were shown further questions specific to searching for source code (relevant to RQ4 and RQ5). The results are shown in Figure 6. Only one participant indicated "Never"; 55 respondents indicated they searched for source code at least some of the time.

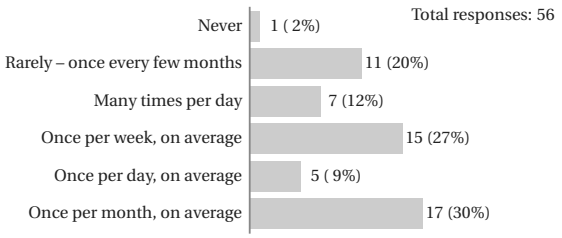

Figure 6: Responses to the question "How often do you search online for software source code?" Predefined answer choices were presented as the mutually-exclusive multiple choices shown on the vertical axis.

4.2. RQ1: How do people look for ready-to-run software?

As mentioned above, after the demographics questions, the remaining questions in the survey questions were only shown if respondents indicated they had a choice in selecting the software they used. All respondents indicated they have some choice in the software they use.

To assess how people located or discovered ready-to-run software, we asked "When you need to find ready-to-run software for a particular task, how do you go about finding software?" The question provided multiple nonexclusive answer choices together with a free-text "Other" option. The predefined answer options were developed based on similar questions posed in other surveys (Bajracharya and Lopes, 2009; Linstead et al., 2009; Sim et al., 2011) and the results of our pilot run. Respondents were free to choose more than one answer. Figure 7 summarizes the results. We separated the responses based on how individuals answered the yes/no question about being involved in software development (Figure 4). The graph is sorted by the sum of responses across developers and nondevelopers for each answer category.

4.3. RQ2: What criteria do people use when choosing ready-to-run software?

We sought to understand the selection and evaluation criteria that may come into play when users try to find ready-to-run software. We posed the question "In general, how important are the following characteristics when you are searching for ready-to-run software for a task?" For the answer options, we provided a two-dimensional grid with different predefined criteria as the rows, and values on a unipolar rating scale for the columns. The available values on the scale were "Rarely or never important", "Somewhat or occasionally important", "Average importance", "Usually of above-average importance", and "Essential". Figure 8 summarizes the results. To indicate which criteria respondents indicated they used, the rows of the graph are sorted by a percentage calculated as the sum of "Essential", "Usually of above-average importance" and "Average importance" ratings divided by the number of possible responses. Not all participants chose to provide a value for every criterion.

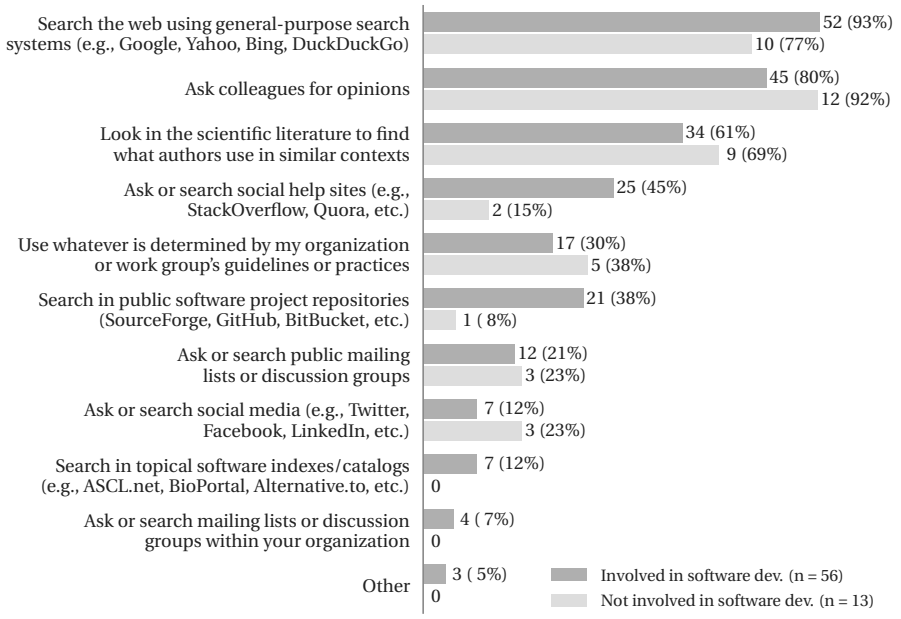

Figure 7: Responses to the question "When you need to find ready-to-run software for a particular task, how do you go about finding software?" Answer choices were nonexclusive. All 69 survey participants answered this question; results are subdivided according to respondents' answers to the question in Figure 4, where 56 people answered "Yes" to involvement in software development and 13 answered "No". Percentages are calculated by subgroup.

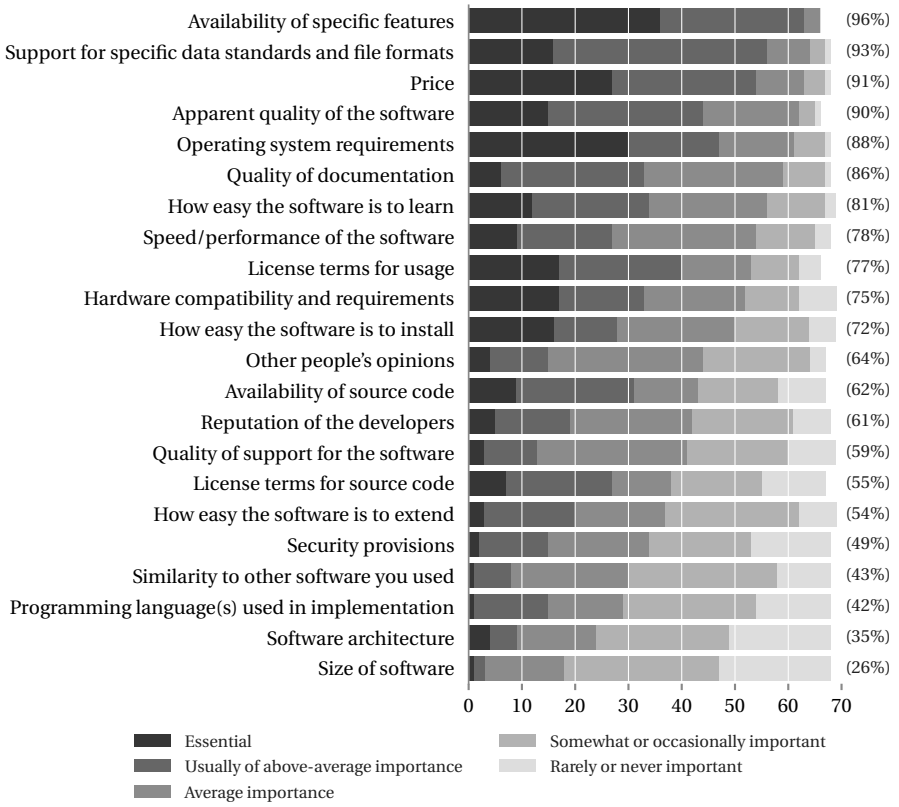

Figure 8: Responses to the question "In general, how important are the following characteristics when you are searching for ready-to-run software for a task?" All 69 respondents answered the question, but not all respondents chose to select an option for every possible characteristic. Responses are ranked by a percentage calculated from the sum of the number of "Essential", "Usually of above-average importance" and "Average importance" ratings for each option divided by the total number of possible responses (69).

4.4. RQ3: What information would people like to find in a catalog of software?

The currently-available catalogs of software are highly heterogeneous in their features and the information they present to users (e.g., Allen et al., 2012; Bönisch et al., 2013; Browne et al., 1995; Gleeson, 2016; Hempel et al., 2016; Hucka et al., 2016; National Aeronautics and Space Administration, 2016; Noy et al., 2009; Shen, 2015). To help inform the development of improved catalogs, we sought to determine what kind of information users find important to provide about software. We posed the following question of all participants who indicated they had the freedom to choose software (not only those who indicated they developed software): "Suppose that it were possible to create a public, searchable catalog or index of software, one that would record information about software of all kinds found anywhere. What kind of information would you find most useful to include for each entry in such a catalog or index?" As with most other questions in our survey, we provided answer choices as nonexclusive multiple choices, with an additional free-text option titled "Other". All 69 participants to our survey replied to this question. Figure 9 summarizes the results. We separated the responses based on how individuals answered the yes/no question about being involved in software development (Figure 4). The graph in Figure 9 is sorted by sum of responses across developers and nondevelopers for each answer category.

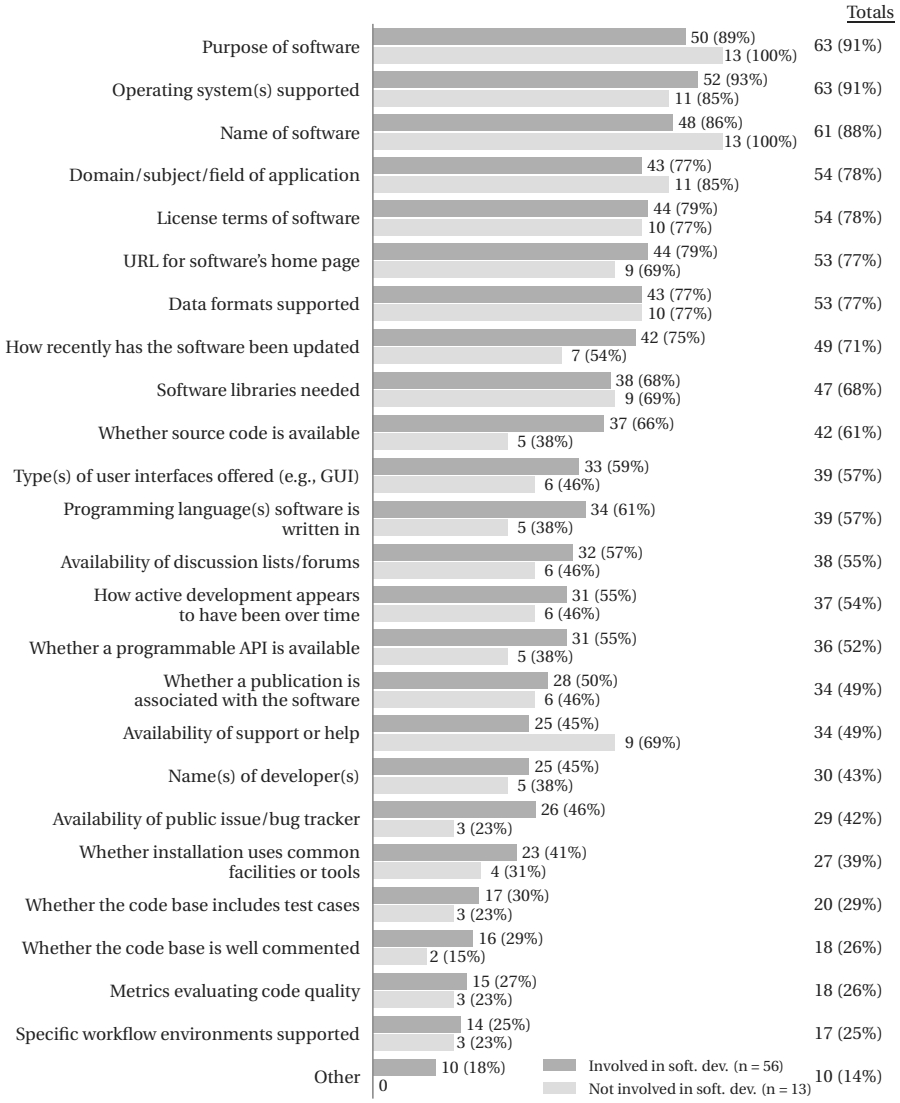

Figure 9: Answers to the question "Suppose that it were possible to create a public, searchable catalog or index of software, one that would record information about software of all kinds found anywhere. What kind of information would you find most useful to include for each entry in such a catalog or index?" There were 24 predefined items and a slot for free text under "Other". Choices were nonexclusive. All 69 survey respondents answered this question; results are shown subdivided according to participants' answers to the question in Figure 4 (left). The graph is sorted by totals; e.g., "Name of software" was the third most selected choice across both developers and nondevelopers.

4.5. RQ4: How do software developers look for source code?

As explained in Section 4.1, the questions concerning searching for source code (RQ4, as well as RQ5 in the next section) were further gated by the question "How often do you search online for software source code?" A total of 55 participants indicated they searched for source code at least some of the time. These participants were shown additional questions, including "What are some approaches you have used to look for source code in the past?". Answer options were nonexclusive multiple choices, including an "Other" option with a field for free-text input. Figure 10 provides a summary of the results. This question was answered by all 55 participants who indicated that they searched for source code at least some of the time (Figure 6).

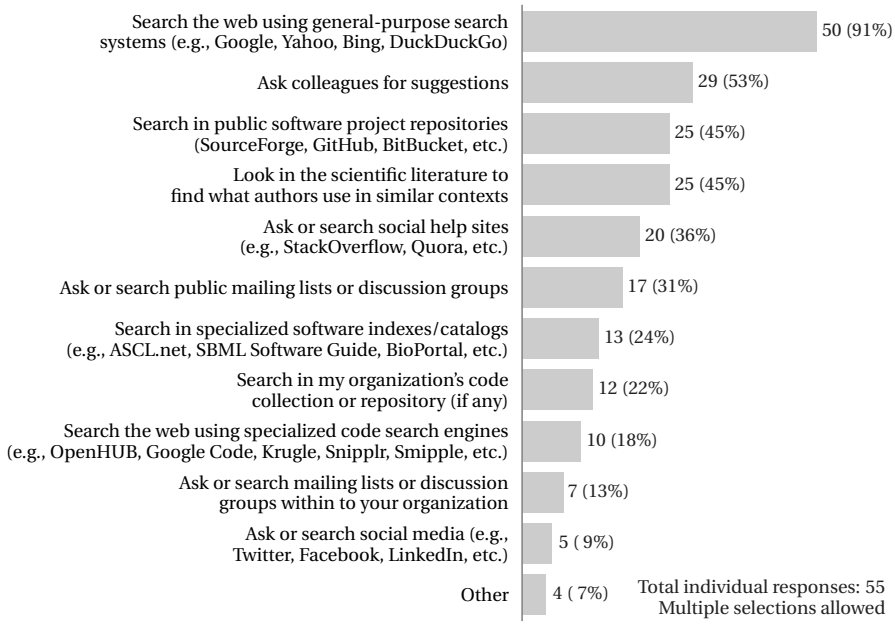

Figure 10: Responses to the question "What are some approaches you have used to look for source code in the past?" This question offered the first eleven predefined categories and an additional slot for free text under "Other". Answer choices were nonexclusive. A total of 55 respondents answered this question.

4.6. RQ5: What can prevent developers from finding suitable source code?

From our own experiences, we know a search for software can fail for a variety of reasons. This motivated our inclusion of another question in the survey: "What are some factors that have hindered your ability to FIND source code in the past?" The question included a variety of nonexclusive predefined options, along with an "Other" option offering a free-text input field. The results are summarized in Figure 11.

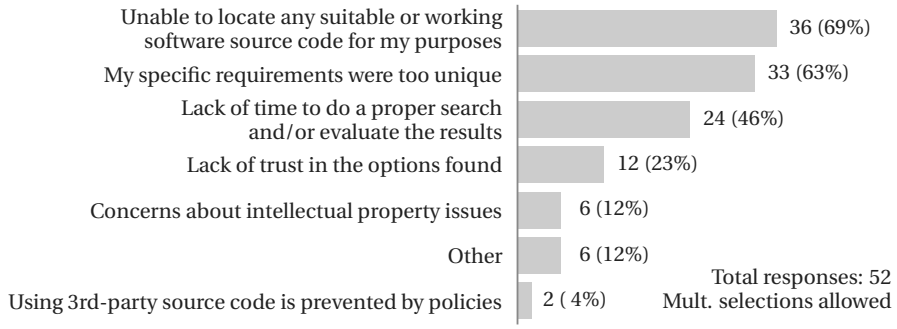

Figure 11: Responses to the question "What are some factors that have hindered your ability to FIND source code in the past?" This question offered the first six predefined categories and an additional slot for free text under "Other". Answer choices were nonexclusive. A total of 52 survey respondents answered this question.

Six out of 52 respondents provided "Other" answers. Three of these were simply explanatory but did not add to the categories listed, two participants cited lack of documentation as a hindrance to either locating or evaluating software, and the third hindrance noted by a respondent was "Some scientific software is hidden from search engines as authors did not bother to put it online or make a small website for it."

5. Discussion

In what follows, we discuss the results in the same order and with the same headings as they were presented in the previous section.

5.1. Demographics

Most of our survey respondents worked in three areas: physical sciences, computing and mathematical sciences, and biology and biological engineering (Figure 1). The responses to the remaining demographic questions (Section 4.1) are consistent with prosaic expectations for the targeted scientific communities. We expected to reach computer-literate individuals, and due to the distribution channels used, we most likely reached those working in research environments. This is similar to other related studies such as that by Lawrence et al. (2015).

Languages such as Python and Java are popular in these settings, and our survey's numbers for languages are consistent with those of a recent Stack Overflow survey (Stack Exchange Inc., 2016) for "most popular technologies per dev type" for their participants who chose "Math & Data". Most of our respondents indicated involvement in software development, and their typical team sizes were small, with 77% being groups of 1 to 5 persons. This is common in scientific software development, especially in academia, and more generally in open-source communities (Sojer and Henkel, 2010). The fact that many respondents indicated they had multiple roles is also consistent—small teams generally require members to take on more than one role.

Among the 81% of the total 69 respondents who were shown the section for software developers, the median number of years of experience was 20 (see Figure 4). This suggests that the typical respondent is mid-career or part of the pre-mobile device generation. Of these, 70% (equal to 56% of the overall 69 respondents) indicated that they were also primarily responsible for project management and/or software architecture, which are traditionally more senior roles. The demographic data may thus indicate a bias in responses against more junior members of the respective communities, such as students and postdocs.

This is worth keeping in mind because junior members may have different search criteria and development experiences than more experienced colleagues. The possibility should be borne in mind when interpreting the survey results in the following sections. The cause of this distribution is unknown.

5.2. RQ1: How do people look for ready-to-run software?

Figure 7 summarizes the responses about how people find ready-to-run software. For both software developers and nondevelopers, the top three choices were (a) using general search engines, (b) asking colleagues and (c) looking in the literature. However, developers differed from nondevelopers in their frequency of selection of these choices: the top choice for developers (93%) was search engines, whereas for nondevelopers, it was personal recommendations (92%). An even larger difference is evident in the use of social help sites such as Stack Overflow, with very few nondevelopers (15%) indicating they used this approach. A similar difference is exhibited with respect to searching public software project repositories such as SourceForge and GitHub, with nondevelopers far less likely to indicate they use that approach. While it is perhaps not surprising that developers would be more familiar with these resources and thus recognizing them as viable options for finding software, the results demonstrate a difference in approaches used by developers versus nondevelopers.

General social media sites such as Facebook appear to be underutilized by both subsets of respondents when searching for ready-to-run software, although nondevelopers were nearly twice as likely to indicate they use these resources as a way to find software. However, both subgroups rarely seem to use domain-specific catalogs. This last result is surprising. A possible explanation is that people may expect general search engines such as Google to index the domain-specific catalogs, and thus, that searching the former will subsume the latter. This does happen in practice: results from at least some of the domain-specific catalogs can easily be demonstrated to show up in Google search outputs, though using the domain catalogs *directly* will usually produce fewer, more relevant results. A second possibility is it reflects a belief that such resources are too

narrowly focused in scope for their needs. A third possibility is that the results reflect ignorance of the existence of topical indexes. Future research should probe this issue further and seek to understand the reasons behind this result.

Finally, the write-in answers for "Other" revealed a category of options we did not anticipate: the use of network-based software package installation systems such as the systems available for the different Linux operating system distributions. In retrospect, this is an oversight in our list of predefined categories—the package management systems do offer some search capabilities, and thus, this is indeed another way for a person to find ready-to-run software. Future surveys should include this as a predefined answer choice.

5.3. RQ2: What criteria do people use when choosing ready-to-run software?

The results of Figure 8 indicate that the most important search criterion is the availability of specific features (96%) in the software. In fact, it was the only characteristic for which none of the respondents chose "Somewhat or occasionally important" or "Rarely or never important". The high ranking of this characteristic is unsurprising: after all, if one is searching for software for a task, paying attention to the software's feature set is paramount. The results also show that support for specific data standards and file formats (93%) and software price (91%) are also major considerations, which may reflect the culture of scientific computing. Operating system requirements (88%) scored highly, as did quality of documentation (86%) and how easy the software is to learn (81%).

How software is implemented in terms of programming language (42%) and the particular software architecture (35%) were deemed relatively unimportant. We speculate that this may be simply because if one is looking for ready-to-run tools, the details of the implementation may not matter as much—it may be that other operational constraints, such as operating system support, may be more pressing constraints than how the software is written.

Quality and support aspects of the software appear to be secondary considerations as far as information to put in a software catalog. Both the reputation of the developer (61%) and the level of software support (59%) were rated in

the middle. A possible explanation for this result is that once the software is installed, users in our sample expect that they can find their own solutions to any possible problems encountered, and that they are less concerned about future software updates. This may reflect the culture of software use in scientific computing; anecdotally, we find these users to be accustomed to less polished software and having lower expectations. By contrast, in consumer or commercial environments, software is often closed-source, licenses are purchased with implied support, and updates are expected (and often issued automatically).

We can compare these results to those of Lawrence et al. (2015) with respect to their question about important factors users consider when adopting new technology. There are notable differences. For example, in their survey, the highest-ranked factor was "documentation available", while in our survey, "quality of documentation" (the closest matching category) ranked sixth overall. Their second-ranked factor, "ability to adapt/customize" is closed to our "how easy the software is to extend", which ranked seventeenth in our survey. While it is true that our survey included many more possible criteria, and in addition, some criteria in Lawrence et al.'s survey question were coarser in detail, many items in both surveys are comparable, so these two differences alone are unlikely to explain the results. We hypothesize two possibilities. First, the context of their survey was scientific computing gateways, whereas our survey was not focused on this and considered people working with any kind of software environment. The contexts may influence the criteria people use. Second, it is possible that the rankings are influenced by the different answer formats: we asked participants to rank the importance of each criterion, while Lawrence et al. asked respondents pick their top three criteria.

Finally, it is interesting to compare these results with those of studies on desirable traits for successful open-source software projects (e.g., Crowston et al., 2003, 2006; Sen et al., 2012; Subramaniam et al., 2009; Tom Lee et al., 2009,?). Intuitively, we expect the criteria should align: the features that people say they use to discriminate between choices when looking for software are presumably the same that differentiate successful software efforts from unsuccessful ones.

And in fact, it does turn out that code quality, documentation quality, price, and licensing terms are recognized indicators of success (Crowston et al., 2003, 2006; Subramaniam et al., 2009; Tom Lee et al., 2009); however, as mentioned above, our survey participants seemed to rate developers' reputations and other people's opinions of software as of middling importance, which does not align with studies about traits of successful software projects.

5.4. RQ3: What information would people like to find in a catalog of software?

Figure 9 reports the results for the question "Suppose that it were possible to create a public, searchable catalog or index of software, one that would record information about software of all kinds found anywhere. What kind of information would you find most useful to include for each entry in such a catalog or index?" When viewed across the subgroups of developers and nondevelopers, the five most-chosen characteristics to include were the operating system(s) supported, the purpose of the software, the name of the software, the domain of application, and license terms—all of which are very basic traits that are logically relevant to anyone looking for software. Nondevelopers particularly favored the name and purpose as their two most-often chosen features to be included in a catalog. (100% of nondevelopers selected them.) After these, the next most-often chosen were a URL for a home page, the data formats supported, and how recently the software was updated.

As might be expected, more developers than nondevelopers chose availability of source code as well as programming language as characteristics they want to see indexed. In other respects, the trend for nondevelopers followed the same pattern as for developers, with one interesting exception: the availability of support or help was ranked by nondevelopers as highly as the home page URL. By comparison, developers chose the availability of bug/issue trackers as often as they chose availability of support or help. A possible explanation that in the minds of developers, support options and issue trackers may be more or less synonymous, whereas nondevelopers may be less inclined to use bug trackers and more inclined to simply contact the software's support address or personnel.

Details about the software, such as the types of user interfaces offered, a programmable API, and the programming language used to implement the software, were of middle importance to survey participants. It came as a surprise, however, that more formal indicators of software development rigor—such as test cases, well-commented code, and metrics evaluating code quality—ranked relatively low, even for developers. We expected developers to be more discerning about the quality of software they choose. A possible explanation is that developers may simply assume they will need to take a closer personal look at any software they choose, so they don't regard it as important to include this information in a software index. This is another aspect of the results that would be worth investigating more deeply in future work.

Finally, ten individuals wrote additional text in the "Other" field of the question. Analysis of these responses revealed that one answer was similar enough to the predefined categories that we included it in the counts shown in the graph, and one response was not interpretable. The remaining write-in values constituted sufficiently different categories of information that they were not truly subsumed by any of the options we provided. The following are the distinct themes that were raised in these responses:

- Price (two mentions)
- Size of the user base (two mentions)
- Availability of documentation (two mentions)
- Size of the software
- Whether it is packaged for Debian
- URL of version control repository
- List of plug-ins available
- List of similar tools
- Stability of parent organization

5.5. RQ4: How do software developers look for source code?

The responses to this question revealed that the use of general search engines was the most popular approach (91%), followed by asking colleagues (53%), and in third place, a tie between consulting the literature and searching in repositories such as SourceForge, GitHub and BitBucket (45% each). This is consistent with findings in some other published studies (e.g., Lawrence et al., 2015), though the relative percentages are different.

The use of specialized software indexes such as ASCL.net ranked much lower (24%), as did searching code collections in one's organization (22%). Code search sites such as Open Hub ranked even lower (18%), and the use of social media systems such as Twitter, Facebook and LinkedIn ranked lower still (9%). Out of the four write-in "Other" answers, one was clearly in the same category as a predefined option, so we adjusted the counts accordingly; the other three were "O'Reilly books", "Look at the web page for that software!" and "What libraries are used by other software that I like?" These last three represent additional approaches not anticipated in our set of predefined answer choices.

These results show that close to half of respondents search project repositories such as GitHub when looking for source code, but unexpectedly, this approach is no more popular than looking in the scientific literature. This may reflect a population sample bias towards researchers in our study: *outside* of research environments, developers may be less likely look in the research literature as often as they search in GitHub. On the other hand, we were surprised at the low ranking of searching topical software indexes.

How do these results compare to those of RQ1, which asked about finding ready-to-run software? Although similar, the two questions were not identical: we offered three different answer choices because the contexts lent themselves to some different actions, and in addition, the question from Figure 7 involved both developers and nondevelopers, whereas *this* question involved *only* developers. Nevertheless, we can compare the common subset of answer categories and the subset of respondents in Figure 7 who identified themselves as software developers. We present the results in Figure 12.

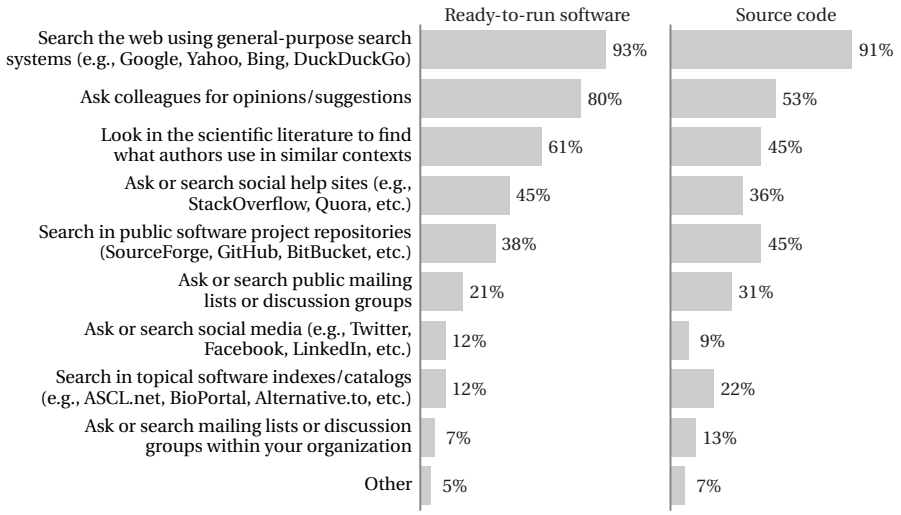

Figure 12: Comparison of the results from 7 and 10 for the overlapping answer categories. (Left) Subset of results from Figure 7 for the 56 respondents who indicated they were involved in software development. The results have been expressed as percentages of the total number of responses for that subgroup of people. (Right) Reproduction of the results of Figure 10.

This shows that the top three approaches for finding *both* ready-to-run software and source code are identical: searching the Web, asking colleagues, and looking in the literature. When looking for source code, searching public repositories such as SourceForge and GitHub rises in popularity; while this is to be expected given the nature of the task and the fact that the respondents were software developers, the approach still only tied with searching the literature.

Some more unexpected results reveal themselves. First, asking colleagues for opinions is far less common when searching for source code than when searching for ready-to-run software (53% versus 80%). We have no hypothesis to explain this difference. A second unexpected result is that the use of social help sites such as Stack Overflow was selected less frequently (36%) when searching for source code than when searching for ready-to-run software (45%). Considering that Stack Overflow is one of the most popular online resources for software developers today, this result is counterintuitive. Two possible explanations present

themselves. First, people may expect that using general-purpose search engines such as Google will return results from Stack Overflow and similar sites, and thus, they may simply not bother searching in the specialized sites directly. A second possible explanation may come from considering responses to the question in the next section below, on the failure of past code searches: people cited an inability to find any code suitable for their purposes (69%) and a belief that the requirements were too unique (63%). Perhaps these experiences and beliefs temper people's expectations of the potential for finding solutions through social help sites. Future surveys should explore this question more deeply.

There is another surprising result: using software catalogs was more popular when the task was searching for source code compared to searching for ready-to-run software. Indexes and catalogs do not offer *source code search* per se, though source code may be available elsewhere (e.g., on the software's home page or repository). Since developers cannot search inside source code from the catalogs, or in most cases even get a sense for the architecture of the software from the catalog descriptions alone, our intuition was that the indexes would be more useful for finding ready-to-run software—the opposite of what the survey revealed. The reasons for this is another opportunity for future research.

Finally, we note that the use of software indexes was still quite low overall (12% in the context of finding ready-to-run software, 22% in the context of source code). It ranked far lower than, for example, searching the literature, despite that software indexes are arguably much better suited to the task of finding software in a given domain or for a given purpose. The same potential explanations we noted in Section 5.2 may apply here: namely, respondents may expect searching in Google will subsume searching in the specialized indexes, or participants may believe the indexes are too narrowly focused, or they may simply not be sufficiently aware of their existence. More generally, this result indicates that the developers of software catalogs continue to face challenges in producing systems that users find sufficiently compelling.

5.6. RQ5: What can prevent developers from finding suitable source code?

The results of our question about search failures (Figure 11) show that the largest hindrance is simply finding a match to one's needs, either because of difficulty finding suitable working software or because none of the options found satisfy requirements. Time limitations also often (46%) impact the ability to conduct proper searches for source code or to evaluate the results. This is may be due to the large number of results that general-purpose search engines can return, which in turn may make it difficult to find suitable results easily. (After all, the ranking systems of general-purpose search engines such as Google are optimizing for a commercial metric, such as potential advertising revenue, rather than metrics more pertinent to searching for software.)

The results also suggest that software licensing (12%) was a minor hindrance, even though it was a relatively important criterion for ready-to-use software (Section 4.3). This suggests that intellectual property information is not sufficiently visible during searches. This is consistent with the format of results presented by Google and similar general-purpose search engines: they do not usually contain license information, unless it happens to be the in the first few words of the text fragment presented as part of a given search result.

Finally, one of the "Other" results written by respondents noted that some software packages lack web pages or other kinds of online presences. This is a notable observation. In effect, it means that the software is hidden from search engines, and may be hidden from search in social coding sites and social media as well. Unfortunately, we do not have data about the types of software in this category. Could it be that these "hidden" software packages are more likely to be older, non-commercial software? After all, commercial efforts are likely to seek to maximize publicity (in order to increase sales), while newer open-source efforts are likely to take advantage of online systems such as GitHub. This is a question that could be probed in future surveys.

Overal, our results are very similar to those of Samadi et al. (2004). For our question about what factors hindered people from finding software (Section 4.6), the most popular reason was finding a match to one's needs.

6. Conclusions

Before the advent of the World Wide Web, even before the current Internet, it was arguably easier to find software for personal computers—there was less of it, and there were simply fewer places to look. Community bulletin boards and archive sites using FTP made software available for copying by anonymous users over telephone networks; later, the Usenet culture (Emerson, 1983) of the 1980's encouraged widespread sharing and even devoted a newsgroup (*comp.sources*) to the exchange of software source code. Communities created manually-curated lists of software (e.g., Boisvert et al., 1985; Brand, 1984) and some journals regularly published published surveys of topical software (e.g., Martinez, 1988). Fast-forward to today, and the staggering wealth of software resources available to users is both a blessing and a curse: one can simultaneously feel that for any given task, "surely *someone* has already written software to do this," and yet an attempt to find suitable software can seem like falling into a rabbit hole.

6.1. How people find software today

So what *do* users do today when they want to find software? This survey was an attempt to gain insight into the approaches used by people working in science and engineering, as well as the criteria that they apply to select between alternative software choices. Our participants worked primarily in the physical, computing, mathematical and biological sciences; the majority were involved in software development and had a mean of 20 years of experience; most worked in small groups; and all had some degree of choice in the software they used. The majority spent over 50% of their day using software; this is somewhat higher than some other studies have reported (e.g., Hannay et al., 2009, found scientists spent 40% of their time using scientific software).

The survey results help identify a number of current community practices in searching for both ready-to-use software and source code:

1. When searching for ready-to-run software (RQ1), the top three approaches overall were are: (i) search the Web with general-purpose search engines,

(ii) ask colleagues, (iii) look in the scientific literature. After these top three, the next most commonly stated approaches differed between those respondents who self-identified as being involved in software development and those did not: more developers indicated asking on social help sites such as Stack Overflow and searching in public software repositories such as GitHub (in that order), while non-developers indicated following their organization's guidelines and a tie between asking on public mailing lists and asking on social media.

2. The top five criteria given above-average weight when searching for ready-to-run software are: (i) availability of specific features, (ii) support for specific data standards and file formats, (iii) price, (iv) apparent quality of the software, and (v) operating system requirements.

3. Regarding information people would like to see in a catalog or index of ready-to-run software, a total of 15 features were indicated as having above-average value by at least 50% of the respondents; of these characteristics, the operating system supported, purpose of software, name of software, domain/field of application, and licensing terms were the five most-often requested features. Developers displayed slightly different preferences compared with nondevelopers, notably with respect to the availability of support or help for a given software product, but on the whole, both subgroups displayed similar preferences.

4. The top five approaches used by developers in science and engineering to search for source code are almost identical to those they use to find ready-to-run software. They are: (i) search the Web with general-purpose search engines, (ii) ask colleagues, (iii) look in the scientific literature, (iv) search in public software project repository sites such as GitHub, and (v) look in social help sites such as Stack Overflow.

5. The top three reasons developers are sometimes *unable* to find source code are: (i) unable to locate suitable software, (ii) requirements are too unique,

and (iii) insufficient time to search or evaluate options. Concerns about intellectual property issues ranked low.

The results above have implications for the development of better resources for locating software. In common with other surveys, we found that more people indicate they use general Web search engines than any other approach for finding both ready-to-run software and source code. This implies that for any specialized resource such as a software catalog to gain popularity, it must be indexed by Google and other search engines so that users can find its content via general Web searches. In addition, the five attributes most important to our respondents when they are seeking software are (i) specific features, (ii) specific data standards supported, (iii) price, (iv) operating system requirements, and (v) apparent quality. This implies that improving people's ability to obtain this information would improve their ability to find software in different situations. Finally, software cataloging efforts would benefit by focusing on the most desirable information items revealed by our survey (Figure 9).

6.2. Lessons for future surveys

Analyzing the survey results has led us to recognize aspects of the survey that could have been improved. First, in the demographic profile questions (Section 4.1), it would have been useful to gather more specific data. For example, the work fields question could have offered finer-grained options, and additional questions could have asked participants about their institutional affiliation (e.g., educational, government, industry) as well as their work roles (e.g., student, staff, faculty). Of course, the benefits of additional questions must be weighed against respondents' patience for filling out long surveys.

Second, the questions asking about software search could have had an explicit answer choice about the use of scientific gateways. The survey questions generally did not mention gateways or portals explicitly; the closest was the question discussed in Figure 9, which included workflow environments as an answer choice. Based on the responses reported in Figure 9, one quarter of the

respondents consider support for workflow environments a criterion in selecting software. Since we did not ask about it explicitly, it is unclear whether any of the participants had the use of gateways in mind and framed their responses accordingly. It is also not clear what effect this would have had on their responses. Gateways concentrate software resources in one location and typically provide an index or other means of finding software provided by the gateway, and it is conceivable that this may change the nature of how users think of finding software or the criteria they use to discriminate between available alternatives. It is therefore possible that this is a confounding factor in our results. Future surveys should address this aspect explicitly.

Third, future work must strive to increase the response rate. While we believe the present survey's results are accurate for the sample of people who finished the survey, we must also acknowledge that a response rate of 3% is disappointing. It is widely asserted that Web-based surveys often encounter low rates (e.g., Couper, 2000; Couper and Miller, 2008; Kitchenham and Pfleeger, 2008); in our experience, many studies even fail to disclose the response rate, or claim a rate without reporting the number of potential recipients, leaving in question the accuracy of the rate. However, of the published surveys that disclose both the number of potential recipients and the number of completed responses received (e.g., Bauer et al., 2014; Kalliamvakou et al., 2014; Lawrence et al., 2015; Sojer, 2010; Wu et al., 2007), the values often have been higher. For example, Sojer (2010) reported 9.7% and Lawrence et al. (2015) obtained 17%, albeit with a highly motivated population. One possible cause for our lower response rate may be the venues where we advertised the survey. Our primary venues for soliciting participation were certain mailing lists and Facebook groups. With respect to the mailing lists, some recipients may not have received the survey messages because automatic spam filters may have blocked the messages from their electronic mail inboxes. This would mean that fewer people saw the invitations than the number of people subscribed to the mailing lists, artificially reducing the apparent response rate. With respect to Facebook, some users may be have signed up long ago but they may rarely or never check the

group we targeted. The latter is especially plausible when we consider two other results of our survey: as shown in Figure 4, respondents had a mean of 20 years of experience, and in Figure 12, social media of Twitter/Facebook/LinkedIn variety were little-used by participants for finding software. If that reflects the overall population we reached and their broader pattern of social media use, then they may simply be of a generation that spends less time on Facebook than a younger generation of researchers. Again, this would cause our estimated number of recipients to be higher than the actual number of people who saw the announcements in that venue. Finally, it is possible that our announcements and/or the front page of the survey were simply not sufficiently motivational.

7. Acknowledgments

We thank Alice Allen, Daniel S. Katz, Sarah M. Keating, Matthias König, Allyson Lister, Rajiv Ramnath, Renee M. Rottner, Lucian P. Smith, and Linda J. Taddeo for many comments and feedback on previous versions of this manuscript. This work was supported by the USA National Science Foundation (award #1533792).

Allen, A., Schmidt, J., Nov. 2015. Looking before leaping: Creating a software registry. Journal of Open Research Software 3 (1).

Allen, A., Teuben, P., Nemiroff, R. J., Shamir, L., Sep. 2012. Practices in code discoverability: Astrophysics Source Code Library. In: Ballester, P. (Ed.), Astronomical Data Analysis Software and Systems XXI. Vol. 461. p. 627.

Bajracharya, S., Lopes, C., 2009. Mining search topics from a code search engine usage log. In: MSR'09: Proceedings of the 6th IEEE International Working Conference on Mining Software Repositories.

Bajracharya, S. K., Lopes, C. V., 2012. Analyzing and mining a code search engine usage log. Empirical Software Engineering 17 (4-5), 424–466.

Banker, R. D., Kauffman, R. J., Zweig, D., 1993. Repository evaluation of software reuse. IEEE Transactions on Software Engineering 19 (4), 379–389.

Bauer, V., Eckhardt, J., Hauptmann, B., Klimek, M., 2014. An exploratory study on reuse at Google. In: Proceedings of the 1st International Workshop on Software Engineering Research and Industrial Practices (SER&IPs 2014). ACM Press, pp. 14–23.

Black Duck Software, Inc., 2016. Black Duck Open Hub. Available on the World Wide Web at `https://www.openhub.net/`. Front page archived on 2017-07-24 at `https://perma.cc/25AM-YQMN`.

Boisvert, R. F., Howe, S. E., Kahaner, D. K., Dec. 1985. GAMS: A framework for the management of scientific software. ACM Transactions on Mathematical Software 11 (4), 313–355.

Bönisch, S., Brickenstein, M., Chrapary, H., Greuel, G.-M., Sperber, W., 2013. swMATH – a new information service for mathematical software. In: Carette, J., Aspinall, D., Lange, C., Sojka, P., Windsteiger, W. (Eds.), Intelligent Computer Mathematics (Held as Part of CICM 2013, Bath, UK, July 8–12, 2013). Lecture Notes in Artificial Intelligence. Springer, pp. 369–373.

Bourne, P. E., Jan. 2015. Foundations for discovery informatics. Available on the World Wide Web at `http://www.slideshare.net/pebourne/foundations-for-discovery-informatics`.

Brand, S. (Ed.), 1984. Whole earth software catalog. Quantum Press/Doubleday.

Brandt, J., Dontcheva, M., Weskamp, M., Klemmer, S. R., 2010. Example-centric programming: integrating web search into the development environment. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. pp. 513–522.

Brandt, J., Guo, P. J., Lewenstein, J., Dontcheva, M., Klemmer, S. R., 2009. Two studies of opportunistic programming: Interleaving web foraging, learning, and writing code. In: Proceedings of the SIGCHI Conference on Human

Factors in Computing Systems. CHI '09. ACM, New York, NY, USA, pp. 1589–1598.

Browne, S., Dongarra, J., Grosse, E., Rowan, T., Sep. 1995. The Netlib mathematical software repository. D-Lib Magazine.

Cannata, N., Merelli, E., Altman, R. B., 2005. Time to organize the bioinformatics resourceome. PLoS Computational Biology 1 (7), e76.

Council, N. R., 2003. Sharing Publication-related Data and Materials: Responsibilities of Authorship in the Life Sciences. National Academies Press.

Couper, M. P., 2000. Web surveys: A review of issues and approaches. Public Opinion Quarterly 64 (4), 464–494.

Couper, M. P., Miller, P. V., Dec. 2008. Web survey methods: Introduction. Public Opinion Quarterly 72 (5), 831–835.

Crook, S. M., Davison, A. P., Plesser, H. E., Jan. 2013. Learning from the past: Approaches for reproducibility in computational neuroscience. In: 20 Years of Computational Neuroscience. Springer, pp. 73–102.

Crowston, K., Annabi, H., Howison, J., 2003. Defining open source software project success. In: Proceedings of the Twenty-Fourth International Conference on Information Systems (ICIS 2003). Association for Information Systems.

Crowston, K., Howison, J., Annabi, H., 2006. Information systems success in free and open source software development: Theory and measures. Software Process: Improvement and Practice 11 (2), 123–148.

Dabbish, L., Stuart, C., Tsay, J., Herbsleb, J., 2012. Social coding in GitHub: transparency and collaboration in an open software repository. In: Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work. pp. 1277–1286.

Emerson, S. L., Oct. 1983. Usenet: A bulletin board for Unix users. Byte 8 (10), 221–236.

Frakes, W. B., Fox, C. J., 1995. Sixteen questions about software reuse. Communications of the ACM 38 (6), 75–87.

Gallardo-Valencia, R. E., Sim, S. E., 2011. What kinds of development problems can be solved by searching the web?: A field study. In: Proceedings of the 3rd International Workshop on Search-Driven Development: Users, Infrastructure, Tools, and Evaluation. pp. 41–44.

Gallardo-Valencia, R. E., Sim, S. E., 2013. Software problems that motivate web searches. In: Finding Source Code on the Web for Remix and Reuse. Springer, Ch. 13, pp. 253–270.

Ge, X., Shepherd, D., Damcvski, K., Murphy-Hill, E., 2014. How developers use multi-recommendation system in local code search. In: Visual Languages and Human-Centric Computing (VL/HCC), 2014 IEEE Symposium on. pp. 69–76.

Gleeson, P., 2016. Current application support for NeuroML. Available on the World Wide Web at `https://www.neuroml.org/tool_support`. Front page archived on 2017-07-24 at `https://perma.cc/W5BU-AYXC`.

Google, Inc., 2006. Google Code Search. No longer available; archived page view available in the Internet Archive at `https://web.archive.org/web/20061010042536/http://www.google.com/codesearch`.

Google, Inc., 2015a. Google Forms. Available on the World Wide Web at `https://www.google.com/forms/about/`, accessed 2015-09-01.

Google, Inc., 2015b. Google Shortener. Available on the World Wide Web at `http://goo.gl`.

Hannay, J. E., MacLeod, C., Singer, J., Langtangen, H. P., Pfahl, D., Wilson, G., 2009. How do scientists develop and use scientific software? In: Proceed-

ings of the 2009 ICSE workshop on Software Engineering for Computational Science and Engineering. SECSE '09. Washington, DC, USA, pp. 1–8.

Hempel, C., Dahan, M., Arnold, C., Dooley, R., Hanlon, M., Lindsey, S., Mock, S., Montoya, D., Rocha, A., Rojas, M., Scarborough, W., 2016. XSEDE user portal: Software search. Available on the World Wide Web at `https://www.xsede.org/software`. Front page archived on 2017-07-24 at `https://perma.cc/HGX4-AK8N`.

Hettrick, S., Dec. 2014. It's impossible to conduct research without software, say 7 out of 10 UK researchers. Available on the World Wide Web at `http://www.software.ac.uk/blog`. Archived at `http://perma.cc/4M65-3WUP`.

Howison, J., Bullard, J., May 2015. Software in the scientific literature: Problems with seeing, finding, and using software mentioned in the biology literature. Journal of the Association for Information Science and Technology.

Howison, J., Deelman, E., McLennan, M. J., Ferreira da Silva, R., Herbsleb, J. D., Jul. 2015. Understanding the scientific software ecosystem and its impact: Current and future measures. Research Evaluation 24 (4), 454–470.

Huang, X., Lu, T., Ding, X., Liu, T., Gu, N., 2013. A provenance-based solution for software selection in scientific software sharing. In: Computer Supported Cooperative Work in Design (CSCWD), 2013 IEEE 17th International Conference on. pp. 172–177.

Hucka, M., Bergmann, F. T., Shapiro, B. E., 2016. SBML Software Guide. Available on the World Wide Web at `http://sbml.org/SBML_Software_Guide`. Front page archived on 2016-05-06 at `http://perma.cc/APR8-A4Z3`.

Hunter, J. D., 2007. Matplotlib: A 2D graphics environment. Computing In Science & Engineering 9 (3), 90–95.

Ince, D. C., Hatton, L., Graham-Cumming, J., Feb. 2012. The case for open computer programs. Nature 482 (7386), 485.

Jansen, B. J., Spink, A., Jan. 2006. How are we searching the world wide web? a comparison of nine search engine transaction logs. Information Processing & Management 42 (1), 248–263.

Johansson, O., Olausson, M., 2016. Alternative.to. Available on the World Wide Web at `http://alternative.to`. Front page archived on 2016-05-06 at `http://perma.cc/U4DL-HPN9`.

Joppa, L. N., McInerny, G., Harper, R., Salido, L., Takeda, K., O'Hara, K., Gavaghan, D., Emmott, S., May 2013. Troubling trends in scientific software use. Science 340 (6134), 814–815.

Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D. M., Damian, D., 2014. The promises and perils of mining GitHub. In: Proceedings of the 11th Working Conference on Mining Software Repositories. pp. 92–101.

Katz, D. S., Feb. 2015. Catalogs and indices for finding (scientific) software. Available on the World Wide Web at `https://danielskatzblog.wordpress.com/2015/02/23/catalogs-and-indices-for-finding-scientific-software/`. Archived at `http://perma.cc/PF3G-3GAZ`.

Katz, D. S., Choi, S.-C. T., Niemeyer, K. E., Hetherington, J., Löffler, F., Gunter, D., Idaszak, R., Brandt, S. R., Miller, M. A., Gesing, S., Jones, N. D., Weber, N., Marru, S., Allen, G., Penzenstadler, B., Venters, C. C., Davis, E., Hwang, L., Todorov, I., Patra, A., de Val-Borro, M., Feb. 2016. Report on the third workshop on sustainable software for science: Practice and experiences (WSSSPE3). Computing Research Repository arXiv:1602.02296.

Katz, D. S., Ramnath, R., Aug. 2015. Looking at software sustainability and productivity challenges from NSF. Computing Research Repository arXiv:1508.03348.

Kitchenham, B. A., Pfleeger, S. L., 2008. Personal opinion surveys. In: Shull,

F., Singer, J., Sjøberg, D. I. (Eds.), Guide to advanced empirical software engineering. Springer-Verlag, Ch. 3, pp. 63–92.

Lawrence, K. A., Wilkins-Diehr, N., Wernert, J. A., Pierce, M., Zentner, M., Marru, S., 2014. Who cares about science gateways?: A large-scale survey of community use and needs. In: Proceedings of the 9th Gateway Computing Environments Workshop. IEEE Press, Piscataway, NJ, USA, pp. 1–4.

Lawrence, K. A., Zentner, M., Wilkins-Diehr, N., Wernert, J. A., Pierce, M., Marru, S., Michael, S., May 2015. Science gateways today and tomorrow: positive perspectives of nearly 5000 members of the research community. Concurrency and Computation: Practice and Experience 27 (16), 4252–4268.

Li, Y., Zhang, L., Xie, B., Sun, J., 2009. Refining component description by leveraging user query logs. Journal of Systems and Software 82 (5), 751–758.

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

Mario, 2016. freshcode.club. Available on the World Wide Web at `https://freshcode.club`. Front page archived on 2016-05-06 at `http://perma.cc/ENH8-GBFV`.

Marshall, J. J., Olding, S. W., Wolfe, R. E., Delnore, V. E., 2006. Software reuse within the earth science community. In: Geoscience and Remote Sensing Symposium, 2006. IGARSS 2006. IEEE International Conference on. pp. 2880–2883.

Martinez, H. M., 1988. Software survey section. Bulletin of Mathematical Biology 50, I–IV.

Mena, E., Illarramendi, A., Royo, J. A., GoñI, A., Sep. 2006. A software retrieval service based on adaptive knowledge-driven agents for wireless environments. ACM Transactions on Autonomous and Adaptive Systems 1 (1), 67–90.

Morin, A., Urban, J., Adams, P., Foster, I., Sali, A., Baker, D., Sliz, P., Apr. 2012. Shining light into black boxes. Science 336, 159–160.

Morisio, M., Ezran, M., Tully, C., 2002. Success and failure factors in software reuse. IEEE Transactions on Software Engineering 28 (4), 340–357.

Murphy-Hill, E., Lee, D. Y., Murphy, G. C., McGrenere, J., Jul. 2015. How do users discover new tools in software development and beyond? Computer Supported Cooperative Work (CSCW) 24 (5), 389–422.

National Aeronautics and Space Administration, 2016. NASA software catalog. Available on the World Wide Web at `https://software.nasa.gov`. Front page archived on 2016-05-06 at `http://perma.cc/FK5E-Q9LF`.

Niemeyer, K. E., Smith, A. M., Katz, D. S., Jan. 2016. The challenge and promise of software citation for credit, identification, discovery, and reuse. Computing Research Repository arXiv:1601.04734.

Noy, N. F., Shah, N. H., Whetzel, P. L., Dai, B., Dorf, M., Griffith, N., Jonquet, C., Rubin, D. L., Storey, M.-A., Chute, C. G., Musen, M. A., Jul. 2009. BioPortal: ontologies and integrated data resources at the click of a mouse. Nucleic Acids Research 37, W170–3.

Orrego, A. S., Mundy, G. E., Jul. 2007. A study of software reuse in NASA legacy systems. Innovations in Systems and Software Engineering 3 (3), 167–180.

Perez, F., Granger, B. E., Hunter, J. D., 2011. Python: an ecosystem for scientific computing. Computing in Science & Engineering 13 (2), 13–21.

Pohthong, A., Budgen, D., 2001. Reuse strategies in software development: an empirical study. Information and Software Technology 43 (9), 561–575.

Poisot, T., 2015. Best publishing practices to improve user confidence in scientific software. Ideas in Ecology and Evolution 8.

Sadowski, C., Stolee, K. T., Elbaum, S., 2015. How developers search for code: a case study. In: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering. pp. 191–201.

Samadi, S., Almaeh, N., Wolfe, R., Olding, S., Isaac, D., 2004. Strategies for enabling software reuse within the earth science community. In: Proceedings of the 2004 IEEE International Geoscience and Remote Sensing Symposium (IGARSS'04). Vol. 3. pp. 2196–2199.

Sen, R., Singh, S. S., Borle, S., Jan. 2012. Open source software success: Measures and analysis. Decision Support Systems 52 (2), 364–372.

Shen, W., Nov. 2015. DARPA open catalog. Available on the World Wide Web at `http://opencatalog.darpa.mil/XDATA.html`. Front page archived on 2016-05-06 at `http://perma.cc/72A5-4FYJ`.

Sherif, K., Vinze, A., 2003. Barriers to adoption of software reuse: a qualitative study. Information & Management 41 (2), 159–175.

Sim, S. E., Agarwala, M., Umarji, M., 2013. A controlled experiment on the process used by developers during internet-scale code search. In: Finding Source Code on the Web for Remix and Reuse. Springer, Ch. 4, pp. 53–77.

Sim, S. E., Alspaugh, T. A., 2011. Getting the whole story: an experience report on analyzing data elicited using the war stories procedure. Empirical Software Engineering 16 (4), 460–486.

Sim, S. E., Gallardo-Valencia, R., Philip, K., Umarji, M., Agarwala, M., Lopes, C. V., Ratanotayanon, S., 2012. Software reuse through methodical component reuse and amethodical snippet remixing. In: Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work. pp. 1361–1370.

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

Singer, J., Lethbridge, T., Vinson, N., Anquetil, N., 1997. An examination of software engineering work practices. In: Proceedings of the 1997 Conference of the Centre for Advanced Studies on Collaborative Research (CASCON '97). pp. 174–188.

Singer, L., Figueira Filho, F., Storey, M.-A. . A., 2014. Software engineering at the speed of light: How developers stay current using twitter. In: Proceedings of the 36th International Conference on Software Engineering. ICSE 2014. ACM, New York, NY, USA, pp. 211–221.

SlashDot Media, 1999. SourceForge.net. Available on the World Wide Web at `http://sourceforge.net`.

SlashDot Media, 2016. SourceForge directory. Available on the World Wide Web at `https://sourceforge.net/directory`. Front page archived on 2017-07-24 at `https://perma.cc/JA2K-3DDD`.

Sojer, M., 2010. Reusing open source code. Ph.D. thesis, Technische Universität München.

Sojer, M., Henkel, J., 2010. Code reuse in open source software development: Quantitative evidence, drivers, and impediments. Journal of the Association for Information Systems 11 (12), 868–901.

Stack Exchange Inc., Mar. 2016. Stack Overflow developer survey results 2016. Available on the World Wide Web at `http://stackoverflow.com/research/developer-survey-2016`. Archived at `http://perma.cc/234X-2K4E`.

Stewart, C. A., Wernert, J., Wernert, E. A., Barnett, W. K., Welch, V., 2013. Initial findings from a study of best practices and models for cyberinfrastructure software sustainability. Computing Research Repository arXiv:1309.1817.

Subramaniam, C., Sen, R., Nelson, M. L., Jan. 2009. Determinants of open source software project success: A longitudinal study. Decision Support Systems 46 (2), 576–585.

Teevan, J., Alvarado, C., Ackerman, M. S., Karger, D. R., 2004. The perfect search engine is not enough: a study of orienteering behavior in directed search. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. CHI '04. ACM, New York, NY, USA, pp. 415–422.

Tom Lee, S.-Y., Kim, H.-W., Gupta, S., 2009. Measuring open source software success. Omega: The International Journal of Management Science 37 (2), 426–438.

Umarji, M., Sim, S. E., 2013. Archetypal internet-scale source code searching. In: Sim, S. E., Gallardo-Valencia, R. E. (Eds.), Finding Source Code on the Web for Remix and Reuse. Springer Verlag, Ch. 3.

Umarji, M., Sim, S. E., Lopes, C., 2008. Archetypal internet-scale source code searching. In: Russo, B., Damiani, E., Hissam, S., Lundell, B., Succi, G. (Eds.), Open Source Development, Communities and Quality. IFIP International Federation for Information Processing. Springer, pp. 257–263.

Van Der Walt, S., Colbert, S. C., Varoquaux, G., 2011. The NumPy array: a structure for efficient numerical computation. Computing in Science & Engineering 13 (2), 22–30.

van Rossum, G., de Boer, J., Dec. 1991. Interactively testing remote servers using the Python programming language. CWI Quarterly 4 (4), 283–303.

Varnell-Sarjeant, J., Amschler Andrews, A., Lucente, J., Stefik, A., May 2015. Comparing development approaches and reuse strategies: An empirical evaluation of developer views from the aerospace industry. Information and Software Technology 61, 71–92.

Völske, M., Braslavski, P., Hagen, M., Lezina, G., Stein, B., 2015. What users ask a search engine: Analyzing one billion russian question queries. In: Pro-

ceedings of the 24th ACM International on Conference on Information and Knowledge Management. pp. 1571–1580.

White, O., Dhar, A., Bonazzi, V., Couch, J., Wellington, C., Oct. 2014. NIH Software Discovery Index meeting report. Available on the World Wide Web at `http://softwarediscoveryindex.org/report/`. Archived at `https://gist.github.com/mhucka/44921ea1e9a01697dbd0591d872b7b22`.

Wilson, G. V., 2006. Where's the real bottleneck in scientific computing? American Scientist 94 (1), 5.

Wu, C.-G., Gerlach, J. H., Young, C. E., Apr. 2007. An empirical analysis of open source software developers motivations and continuance intentions. Information & Management 44 (3), 253–262.

Xia, X., Bao, L., Lo, D., Kochhar, P. S., Hassan, A. E., Xing, Z., Apr. 2017. What do developers search for on the web? Empirical Software Engineering.