Software search is not a science, even among scientists

Michael Hucka and Matthew J. Graham
mhucka@caltech.edu, mjg@caltech.edu
California Institute of Technology
Pasadena, CA 91125, USA

May 7, 2016

Abstract

When they seek software for a task, how do people go about finding it? Past research found that searching the Web, asking colleagues, and reading papers have been the predominant approaches—but is it still true today, given the popularity of Facebook, Stack Overflow, GitHub, and similar sites? In addition, when users *do* look for software, what criteria do they use? And finally, if resources such as improved software catalogs were to be developed, what kind of information would people want in them? These questions motivated our cross-sectional survey of scientists and engineers. We sought to understand the practices and experiences of people looking for ready-to-run software as well as people looking for source code. The results show that even in our highly educated sample of people, the relatively unsophisticated approaches of relying on general Web searches, the opinions of colleagues, and the literature remain the most popular approaches overall. However, software developers are 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. We also found that when searching for source code, poor documentation was the most common reason for being unable to reuse the code found. Our results also reveal a variety of characteristics that matter to people searching for software, and thus can inform the development of future resources to help people find software more effectively.

1. Introduction

Software is critical to research [30, 32–34, 38, 44, 45, 54, 80, 91], yet finding software suitable for a given purpose remains surprisingly difficult [9, 13, 33, 90]. Few resources exist to help users discover options or understand the differences between them [90]. A recent study [6] of developers at 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*.

When asked, many people say they look for software by searching the Web with a general-purpose search engine such as Google [64, 85]. Despite its popularity, this approach suffers from significant problems: Web search can yield dozens of viable candidates—and millions of irrelevant results. Moreover, some questions cannot be answered through Web searches without substantial additional effort, such as what are the *specific* characteristics of each software tool or how do tools *differ* from each other. Many researchers also look in the scientific literature to learn what others have used for similar tasks [41, 48]. Searching the literature can produce more relevant results and provide other information, but it suffers from limitations too: publications are static documents that may not reflect a tool's current capabilities [92], and moreover, not all publications describe the software they use [33]. This can happen for various reasons, such as article length limits, an expectation that software will be described in other publications, or a disinclination by researchers to describe their full software stack or workflow. Still other methods for finding software include asking colleagues, asking on social media, searching scientific computing gateways, and more.

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 [13, 16, 57, 62, 65, 90]. We are interested in developing better systems to help users, especially scientific users, locate software for their needs. In order to understand the factors that influence how software users locate software, in 2015 we conducted an electronic survey involving members of numerous mailing lists in astronomy and systems biology. Here, we report on our methods, the survey responses, and our findings.

2. Survey design

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.

2.1. Instrument development

We developed the survey instrument iteratively. We began with an initial version in which we posed many questions related to searching for software. Following the practices of other surveys in computing [e.g., 46, 88], we designed the instrument iteratively and payed 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.

To help iterate on the design of the survey instrument, 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 in Appendix A. 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 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.

2.2. Administration

We used Google Forms [29] to implement the survey instrument. The version of Google Forms was the free edition made available by Google, Inc., 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 [28], 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. Sample invitation letters are included in Appendix B. The survey had no express closing date.

2.3. 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.

2.4. 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. We revisit this topic in Section 8.

2.5. Analysis

Simple descriptive statistics were performed using custom programs written in the language Python [60, 87], version 3.4, in combination with the NumPy [86] package, version 1.10.4. The figures in this paper were generated with the help of the Python library Matplotlib [37], version 1.5.1.

3. Results: demographics

The survey included several questions to gather general demographic information about the respondents. One of the first questions in the survey 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 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.

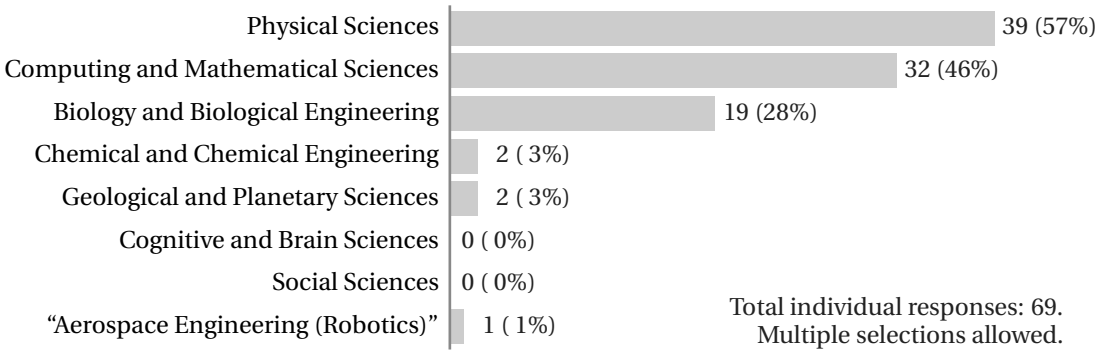

Figure 1: Respondents by discipline. The survey offered the first eight predefined categories and an additional slot for write-in, free-text answers. Choices were nonexclusive. Some respondents included write-in answers but the answers were subsumed by one of the predefined categories; in those cases,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

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 that the overwhelming majority of our respondents spend 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.

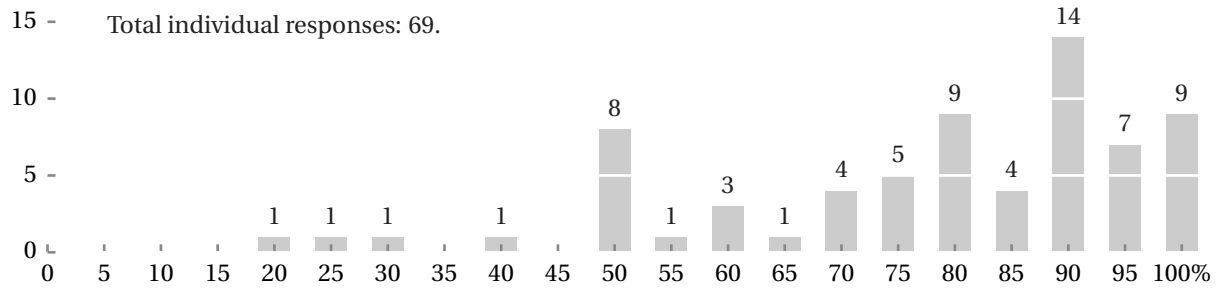

Figure 2: Bar graph of responses to the question "In your work, on a typical day, approximately what fraction of your time involves using or interacting directly with software on a computer or other computing device?"

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 would not be meaningful.) This consideration 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 feedback page. Figure 3 provides the results for this question. It shows that every one of our respondents had some choice in the software they used, and consequently all 69 respondents were shown the next set of questions in the survey.

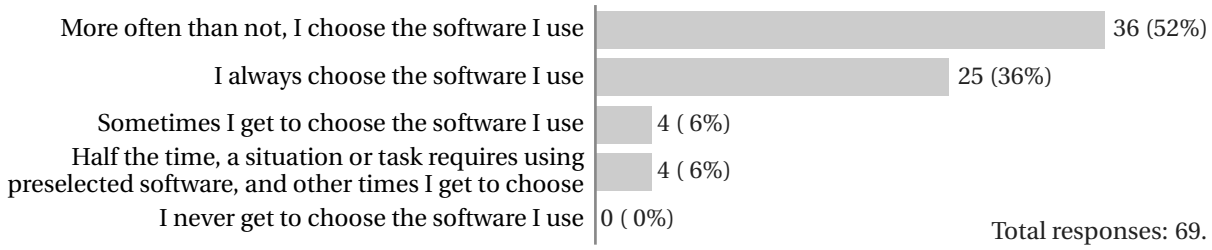

Figure 3: Responses to "In your work, how much freedom do you usually have to choose the software you use?".

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 an additional set of 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 demographic 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).

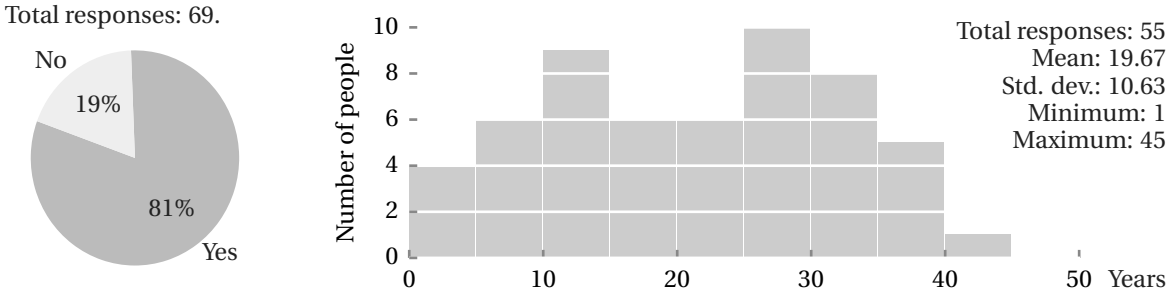

Figure 4: (Left) Responses to the question "Are you involved in software development?" (Right) Histogram plot of years that respondents have been developing software (for those who also answered "Yes" to the question of whether they were involved in software development).

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 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 on the following page provides a tally of the responses.

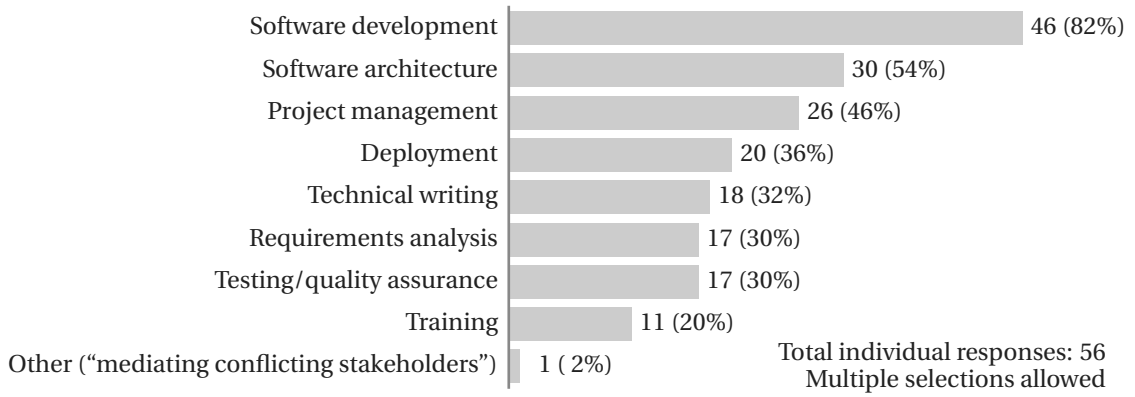

Figure 5: Responses to the question "In your current (or most recent) software development project, what is (or was) your primary responsibility?" This question was shown only to the 56 respondents who answered "Yes" to the question of whether they were involved in software development. This survey question 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.

We also asked, "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 on the previous page), 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)".

In the final demographic question, we asked "Which programming and/or scripting language(s) have you had the most experience with?" This question 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%).

These responses are consistent with expectations for the targeted scientific communities. We expected to reach computer-literate individuals, and due to the distribution channels we used, most likely reached those working in research environments. Languages such as Python and Java are very popular in those settings, and our survey's numbers for languages are consistent with those of a recent Stack Overflow survey [79] for "most popular technologies per dev type" for their participants who chose "Math & Data". Most respondents indicated they are involved in software development, and their typical development team sizes were small, with 77% being in groups of 1 to 5 persons. This is common in scientific software development, especially in academia, and 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 this section, the median number of years of experience was 20. 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 concerning 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. We speculate that it may be due to a degree of self-selection, in that more experienced individuals are more likely to participate in community surveys. In any case, the possible experience bias is something that future similar efforts should aim to redress.

4. Results: how respondents find software

The presentation of the questions in this section was triggered by a screening question asking people if they had a choice in selecting the software they used. All respondents in our sample indicated they have some choice in the software they use, and thus all participants were shown the questions in this section.

4.1. Ready-to-use software

We first consider the search for software for a particular task rather than for development purposes. This does not imply searching for source code, and our questions emphasized that "ready-to-run" was the primary goal. Note it is plausible that source code availability is still a consideration, and in the second subsection below (Section 4.1.2 on the following page), we seek to expose how important this criterion is.

4.1.1. Approaches

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 our own experiences as well as similar questions posed in other surveys [3, 50, 71] and the results of our pilot run. Respondents were free to choose more than one answer. Figure 6 summarizes the results. We separated the responses based on how individuals answered the yes/no question about being involved in software development (Section 3). The graph is sorted by the sum of responses across developers and nondevelopers for each answer category.

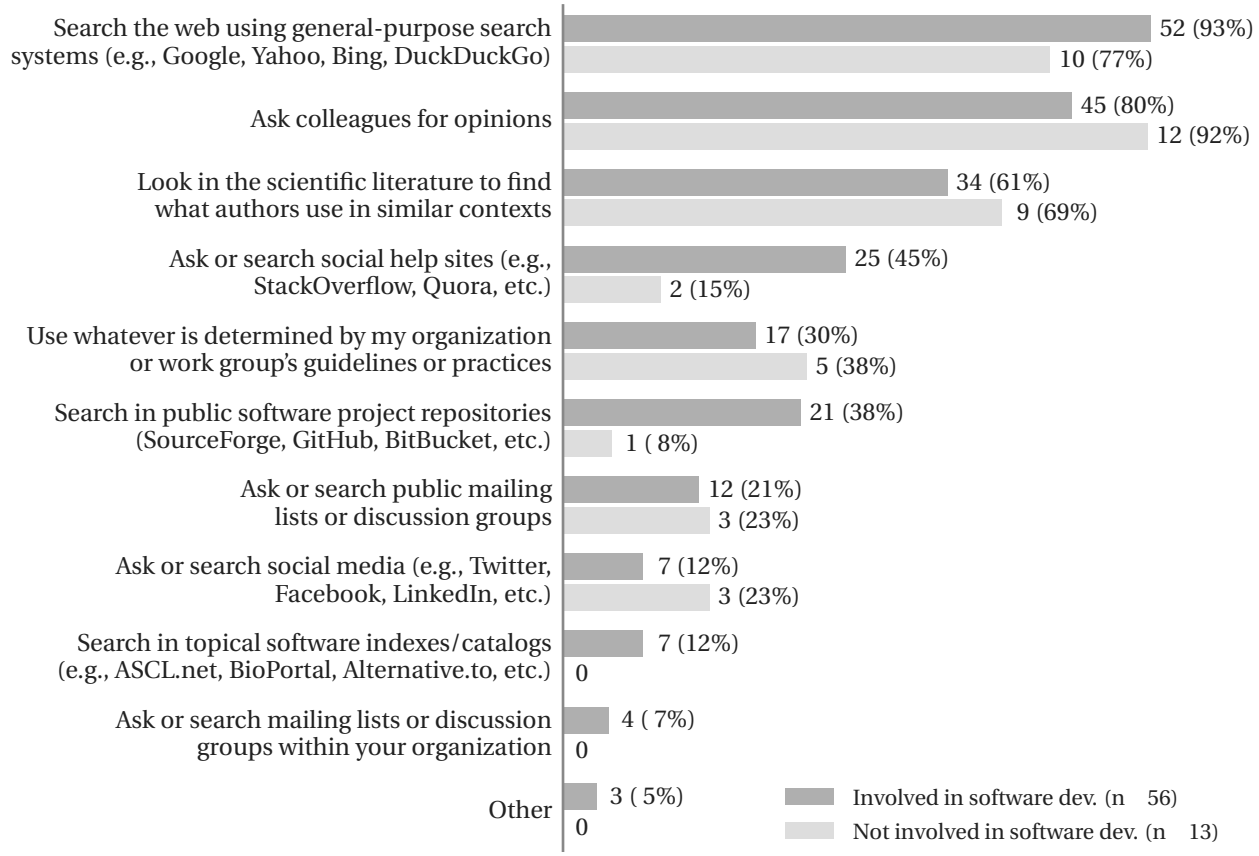

Figure 6: Responses to the question "When you need to find ready-to-run software for a particular task, how do you go about finding software?" Answer choices were nonexclusive. All 69 survey participants answered this question; results are subdivided according to respondents' answers to the question in Figure 4 on page 5, where 56 people answered "Yes" to involvement in software development and 13 answered "No". Percentages are calculated by subgroup.

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 nevertheless indicate 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: all of the answers concerned the use of network-based software package installation systems such as MacPorts [22] and the systems available for the different Linux operating system distributions. In retrospect, this is an obvious oversight in our list of predefined categories—the package management systems offer 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.

4.1.2. Criteria

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 7 on the following page summarizes the results. The rows of the graph are sorted using the sum of "Essential" and "Usually of above-average importance" ratings for each criterion, to reveal the strongest preferences. Not all participants provided a value for every criterion, which may be due either to oversight (e.g., if they did not notice they missed a row in the grid) or confusion about the instructions (if they thought they should only rate the ones they cared about). In the discussion below, percentages in parentheses are calculated as the sum of the number of "Essential" and "Usually of above-average importance" ratings an answer received divided by the total number of responses to the question (69).

The results show that the most important search criterion is the availability of specific features (91%) 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 show that support for specific data standards and file formats (81%) and software price (78%) are also major considerations, which may reflect the culture of scientific computing—software often is expected to be free, and specific areas of science often use specialized data

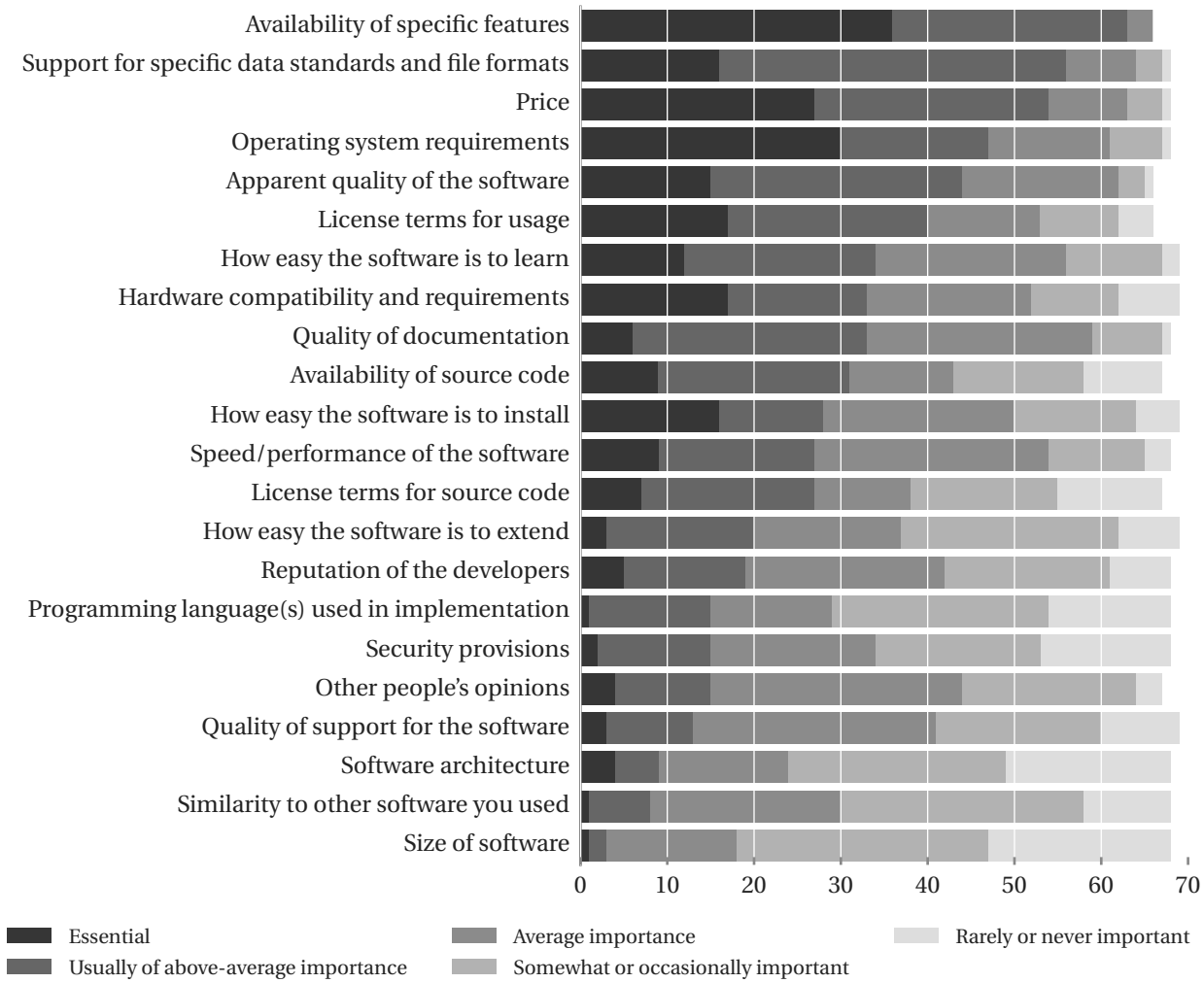

Figure 7: Responses to the question "In general, how important are the following characteristics when you are searching for ready-to-run software for a task?" All 69 respondents answered the question, but not all respondents chose to select an option for every possible characteristic. The bar graph is sorted by the sum of the number of times the options "Essential" and "Usually of above-average importance" were selected for each characteristic.

formats (e.g, FITS in astronomy). Operating system requirements (68%) scored highly, while ease of usage (49%), hardware requirements (48%) and quality of documentation (48%) were all ranked roughly equally. Performance considerations (39%) had only middling impact.

How software is implemented in terms of programming language (22%) and the particular software architecture (13%) were deemed relatively unimportant. 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 are also secondary considerations as far as information to put in a software catalog. Both the reputation of the developer (28%) and the level of software support (19%) rated relatively low. This suggests 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 likely reflects the culture of software use in scientific computing and how it differs from consumer or commercial environments; in the latter, software is often closed-source, licenses are purchased with implied support, and updates are expected (and often issued automatically).

A more surprising result is that other people's opinions of the software (22%) and similarity to other software (12%) did not rank higher. It is possible that users resort to searching for software only when they lack other means of making selections, such as no word-of-mouth recommendations. In those cases, the approval rating of the software or its apparent familiarity may be largely ignored in favor of other criteria. Future surveys or interviews could investigate these points more deeply.

Another surprising result is that, even though the question explicitly concerned ready-to-use software, the availability of the source code (45%) and its licensing (39%) were still relatively important considerations. This may reflect our particular population: a large fraction of responders identified themselves as developers. Such users are more likely to be capable and willing to alter the software to meet their specific requirements; thus, paying attention to source code and its licensing may be important for them.

It is interesting to compare these results with those of software engineering studies on desirable traits for successful open-source software projects [e.g., 17, 18, 66, 81, 83, 83]. 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 [17, 18, 81, 83]; however, as mentioned above, our survey participants seem relatively unconcerned with developers' reputations or other people's opinions of software, which does not align with studies about traits of successful software projects.

4.1.3. Search case histories

To explore more deeply the topic of how users find software, we sought examples of participants' past experiences by asking an optional open-ended question: "Please describe a past scenario when you looked for ready-to-run software." The survey form provided a text editing field where participants could write their responses in free-form text. We received a total of 23 responses, of which 14 contained substantial details about the procedures or steps followed. Figure 8 on the next page provides three examples taken from among those 14 responses.

Analysis of all responses to this question showed that the work tasks discussed by people covered a wide range of applications, ranging from authorization management software, to statistical computing software, to visualization packages. However, the actual process described by different people was often essentially the same: firstly a broad search, usually using Google, with three to four keywords relating to functionality, implementation, and particular formats or standards, if required: e.g. "fits viewer windows" or "grib format Linux". Recommendations from colleagues can also replace the initial broad search phase. The results were reviewed on the basis of the brief descriptions returned and if one obviously met the search context then the link was followed. A typical search would follow four to five such links, and subsequent review criteria were then used to compare these. More recent software is definitely favored but many final decisions are only made when different packages have been installed and compared at a functional or operational level: "I like to develop my own critical view by testing extensively the software."

This anecdotal evidence is consistent with the responses to other questions in our survey. Online searches (for ready-to-use software) are primarily used to identify an initial set of candidate packages that meet a particular set of broad criteria for subsequent (offline) evaluation rather than resulting in the trusted identification of a specific match to a sophisticated query. Although this may reflect user bias, it also points to a lack of functionality in general search engines: one participant wrote "I really miss freshmeat.net [*a now-defunct software index*] because it allowed you to search for software with particular tags/keywords with particular license and language requirements. I haven't seen a good replacement for that."

Interestingly, the reputation of the organization producing the software can result in a level of trust in the search results: "Saw the first match ... [*and*] trusted it because their employees and contractors are a large portion of the folks worldwide who work with that data format." The final match also does not have to be a perfect solution, as indicated by comments made in the case descriptions such as "not really happy

— Sample A —

"Looked for a free UML modeler
Googling for some software comparison pages
Tried a few one (free or with demo license)
Kept ArgoULM
Not really happy with it"

— Sample B —

"Recently I needed to find a package that would let me generate uuids in a specific language. I wanted this to be simple (it wasn't the main point of the project, so I didn't want to reinvent-the-wheel), it ideally needed to be cross platform (testing on mac, running at scale on linux), and it needed to be something that I could install relatively quickly.

One large constraint was I needed to find a package that worked even on relatively out-of-date systems. This meant the newer system-packaged libraries weren't available, and trying to build them from source wasn't all that tractable (I quickly was going down a rabbit hole of other dependencies which weren't available on this old system). So it was very difficult to find a relatively self-contained, especially when I didn't have root access.

My approach was to first check with package managers. That wasn't terribly helpful, without having root access. My next attempt was to find what would have been included on a new system (found through Googling, StackOverflow posts, man pages, etc). Then finally, I had to start searching for older versions, which were more standalone. This final process of searching for older packages was much more random-walk googling + trial-and-error. This last stage was probably the biggest pain.

(Ultimately I was able to find some software that did what I needed.)"

— Sample C —

"Looking for an authorization management software (not authentication).
I did some online search, plus got some info from a conference and colleagues developing one.
The search went on by some specific requirement about the authorization data model used by software and the interfaces available to the authorization data base.
Still no exact solution found.
One software was too complex with respect with the tasks we need and seemed to miss a top requirement (still investigating).
Another seems simpler but definitely lacks documentation on how to start using it."

Figure 8: Samples of responses received to the question "(Optional) Please describe a past scenario when you looked for ready-to-run software." A total of 23 survey respondents answered this question.

with it" to "works well enough". This suggests a degree of pragmatism that workarounds will be found for minor issues with the identified software (see above).

4.2. Source code

As described in Section 3, the survey included the question "Are you involved in software development?" Responses to that yes/no question controlled the presentation of additional questions relevant to developers. In that additional group of questions, one question was "How often do you search online for software source code?" with six answer choices that included "Never". If respondents chose any option *other* than "Never", they were shown further questions specific to searching for source code. In this section, we discuss the results of that part of the survey.

4.2.1. Motivations

Out of 56 participants who answered "Yes" to whether they were involved in software development, a total of 55 (98% of that subset) indicated they searched for source code at least some of the time. Figure 9 summarizes the responses received to the question "How often do you search online for software source code?" To help understand people's motivations for searching for software, we also asked another question: "What are some reasons why you look for source code (when you do look)?" Similar to other questions in the survey, it included multiple nonexclusive choices and a free-text "Other" field as the answer options. Figure 10 summarizes the responses.

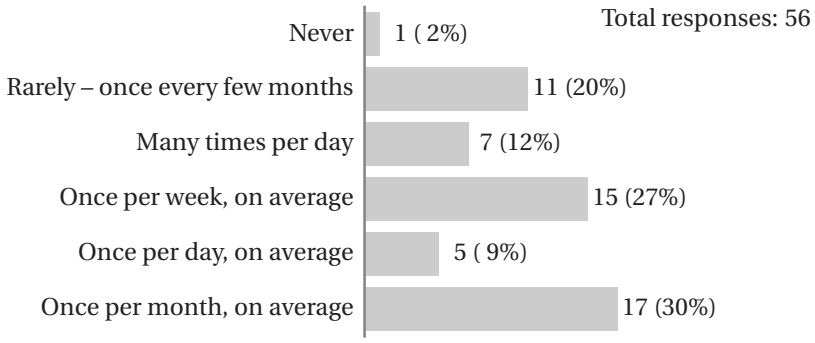

Figure 9: Responses to the question "How often do you search online for software source code?" Predefined answer choices were presented as the mutually-exclusive multiple choices shown on the vertical axis.

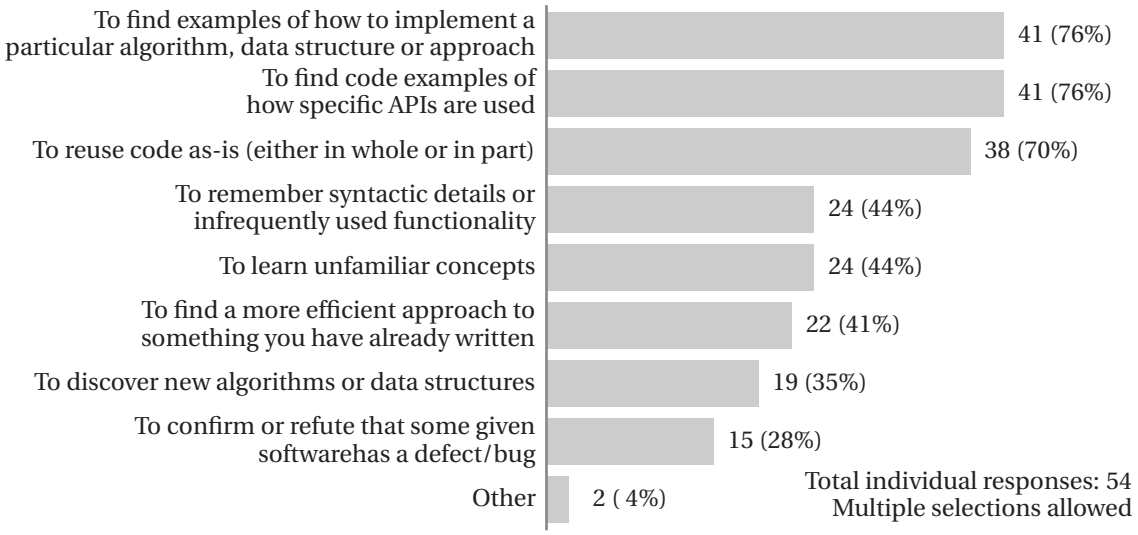

Figure 10: Responses to the question "What are some reasons why you look for source code (when you do look)?" This question offered the first eight predefined categories and an additional slot for free text under "Other". Answer choices were nonexclusive. The "Other" results are discussed in the text.

The results show that one in five of the developers who answered this question (12% + 9% = 21%) are very frequent searchers, performing searches one or more times per day. At the opposite end of the spectrum, one in five searched for source code relatively infrequently, roughly once every few months (20%).

Seeking code examples (76% for each of two questions) and reusing existing code as-is (70%) were given as frequent motivations for searching. This indicates a preference for not reinventing the wheel or, at least, a reluctance to start from an absolutely blank slate when developing. The survey revealed that software

refactoring is another common reason for searching, either to find a more efficient approach to an existing task (41%) or to discover new algorithms and data structures (35%). The Web is also used as an online reference source to recall syntactic details (44%) or learn unfamiliar concepts (44%).

Two respondents wrote answers in the "Other" field: "Find software libraries to use", and "To understand in detail behaviour of software I'm using". Neither are subsumed exactly by other answer options, and indicate additional uses for software source code search that we had not anticipated. The former answer suggests that examining other developers' code can lead to the discovery of previously unknown software libraries; this is different from searching for how specific APIs are used because it does not presuppose knowing which API library will be found or used. The second answer suggests that another use of searching for source code is to understand detailed properties of some software, which is different from seeking to reuse code, learning how to use a specific API, or learning about unfamiliar code concepts.

4.2.2. Approaches

The question "What are some approaches you have used to look for source code in the past?" concerned the methods used by developers to find source code. Figure 11 provides a summary of the results. Answer options were nonexclusive multiple choices, including an "Other" option with a field for free-text input. This question was answered by all 55 participants who indicated that they searched for source code at least some of the time (Figure 9 on the previous page).

The responses to this question revealed that the use of general search engines was the most popular approach (91%), followed by consulting colleagues (53%), and in third place, a tie between consulting the literature and searching in repositories such as SourceForge, GitHub and BitBucket (45% each). The use of more specialized software indexes such as ASCL.net ranked much lower (24%), as did searching the code collections of one's own 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%).

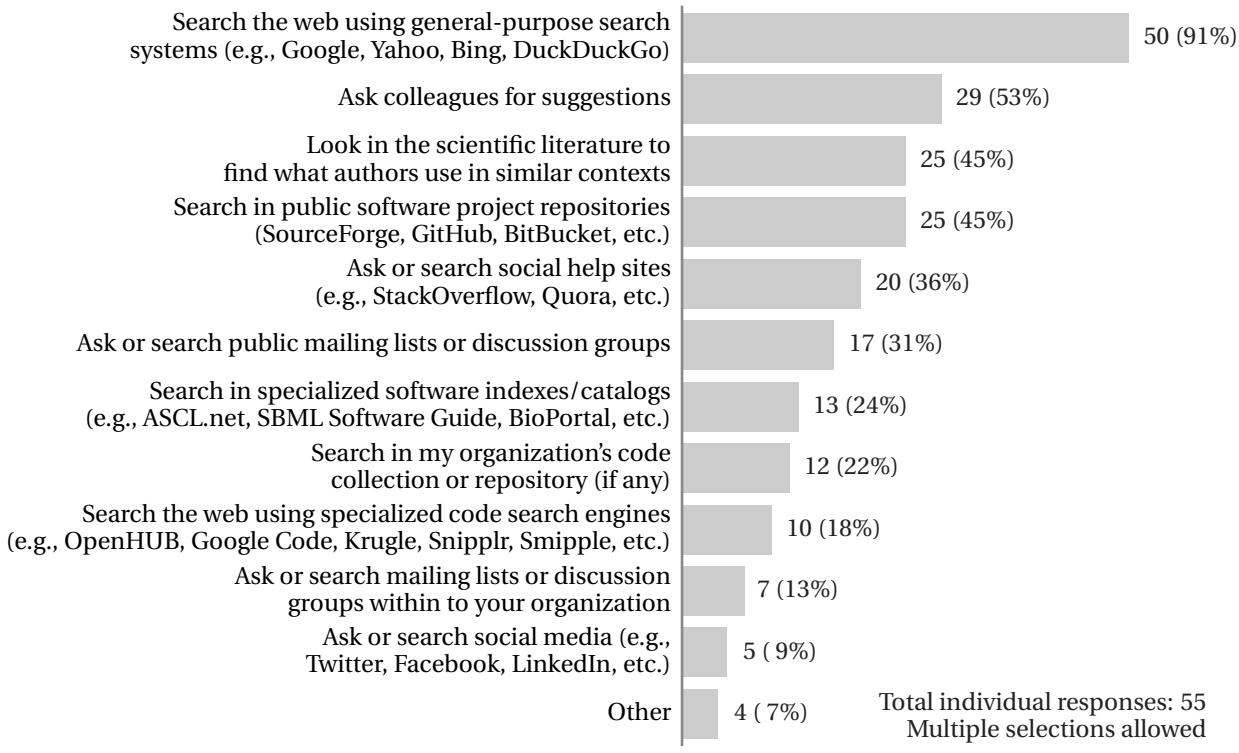

Figure 11: Responses to the question "What are some approaches you have used to look for source code in the past?" This question offered the first eleven predefined categories and an additional slot for free text under "Other". Answer choices were nonexclusive. A total of 55 respondents answered this question.

Out of the four write-in "Other" answers, one was clearly in the same category as one of the predefined options, so we adjusted the count accordingly; the other three answers 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: we suspect that developers *outside* of research environments are unlikely to look in the research literature as often as they search in GitHub. On the other hand, we were surprised again at the low ranking of searching topical software indexes.

How do these results compare to those of Section 4.1.1, 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 6 on page 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 6 who identified themselves as software developers. We present the results in Figure 12.

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. It is possible that this is again a result of our population sample consisting mainly of scientists and engineers.

Some more unexpected results reveal themselves. First, asking colleagues for opinions is far less common when searching for source code than when searching fo ready-to-run software (53% versus 80%). We have

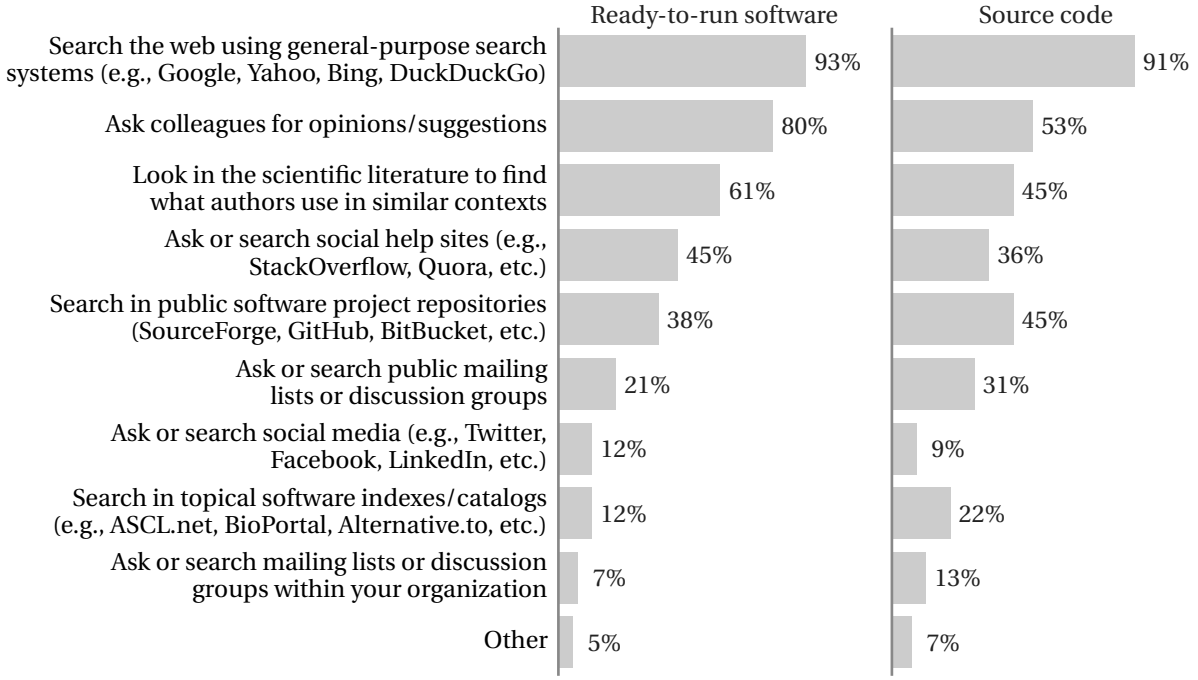

Figure 12: Comparison of the results from Figure 6 and Figure 11 for the overlapping answer categories. (Left) Subset of results from Figure 6 on page 7 for the 56 respondents who indicated they were involved in software development. The results have been expressed as percentages of the total number of responses for that subgroup of people. (Right) Reproduction of the results of Figure 11 on the previous page.

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.

The results also show another surprising result: the use of software catalogs such as ASCL.net and BioPortal was more popular when the task was searching for source code compared to searching for ready-to-run software. This is puzzling because the indexes do not offer source code search per se; therefore, our intuition was that the indexes would be more useful for finding ready-to-run software—but that is the opposite of what the survey revealed. The answer must therefore lie in some other aspect of their features or their use in the context of finding source code. This is another area for future research to explore further.

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 4.1.1 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.

4.2.3. Reasons for *search* failures

The inability to find suitable source code would hinder software reuse. 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 13.

The results 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

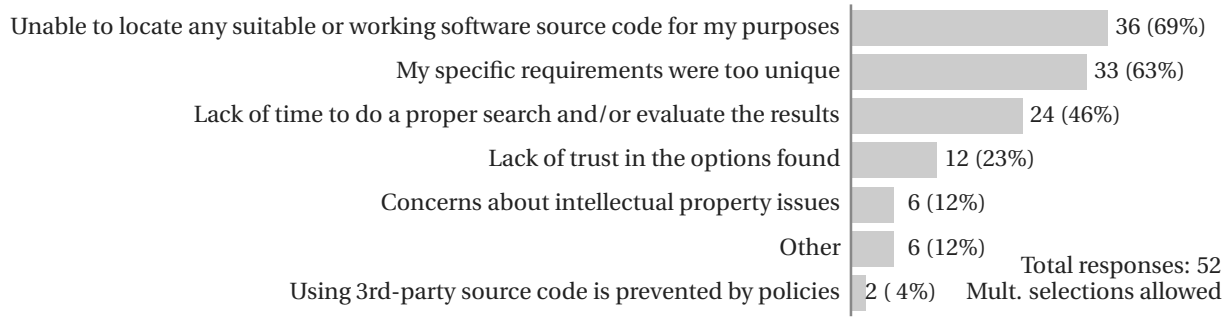

Figure 13: Responses to the question "What are some factors that have hindered your ability to FIND source code in the past?" This question offered the first six predefined categories and an additional slot for free text under "Other". Answer choices were nonexclusive. A total of 51 survey respondents answered this question.

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 to previous successful source code searches, even though it was a relatively important criterion for ready-to-use software (Section 4.1.1), but it was a large factor (48%) in not actually reusing the source code found (Section 4.2.4). 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.

The write-in answers for the "Other" field provided additional insights into factors that contribute to being unable to locate software. Six respondents provided "Other" answers; three of these were explanatory and useful in this context. Two participants cited lack of documentation as a hindrance to either locating or evaluating software. 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."

4.2.4. Reasons for *reuse* failures

Being able to find software source code is not the only factor determining whether developers ultimately make use of the code found; other factors can hinder its use. This motivated our survey question "If you searched and found source code in the past, what are some factors that may have prevented you from REUSING the source code you found?" As with most of the other questions in this survey, this one took the form of a multiple choice question with nonexclusive answer options and a free-text "Other" field. We developed the multiple choice options based on our own personal experiences and feedback from the pilot survey. Figure 14 on the following page summarizes the results.

The results show that the quality of the documentation (62%) and implementation details—language (60%) and operating system (50%)—are the prime factors in determining whether found software will be reused. An inability to compile (44%) the source code also appears to be a relatively common problem, although it is unclear whether this is due to issues with third-party dependencies, compiler versioning, or the actual source code itself. Other common hindrances are discrepancies between what code purports to do (and so matches a search) and whether it actually *can*: this seems to be more common with functionality (38%) aspects than support for particular standards (15%).

The actual quality (35%) of the code itself, its structure (33%), and the design of any algorithm (29%) are less significant factors. Performance (21%) and verifiability (8%) are also surprisingly minor considerations to reuse, but this may be related to the didactic nature of many searches: it may not actually be that important if the code is efficient (or even fully functional) if a developer is only looking to learn something. Finally, the level of support (23%) for any code remains a minor factor and pricing (31%) is also a less significant consideration than for ready-to-run software. Once code has been identified that actually meets the search parameters, it seems that developers are more willing to pay for it (presumably there may be some way to pass the costs on to end-users).

5. Results: information desired about software

A potential aid to finding software is a software catalog or index that classifies known software and allows people to browse and search by various criteria [1, 43, 52, 53, 90]. There exists a number of public software catalogs today; most such resources are domain/community-specific [e.g., 2, 8, 12, 26, 31, 36, 56, 58, 67], though some general indexes also exist [e.g., 7, 40, 51, 77]. The currently-available catalogs are highly heterogeneous in their features and the information they present to users. To help inform the development

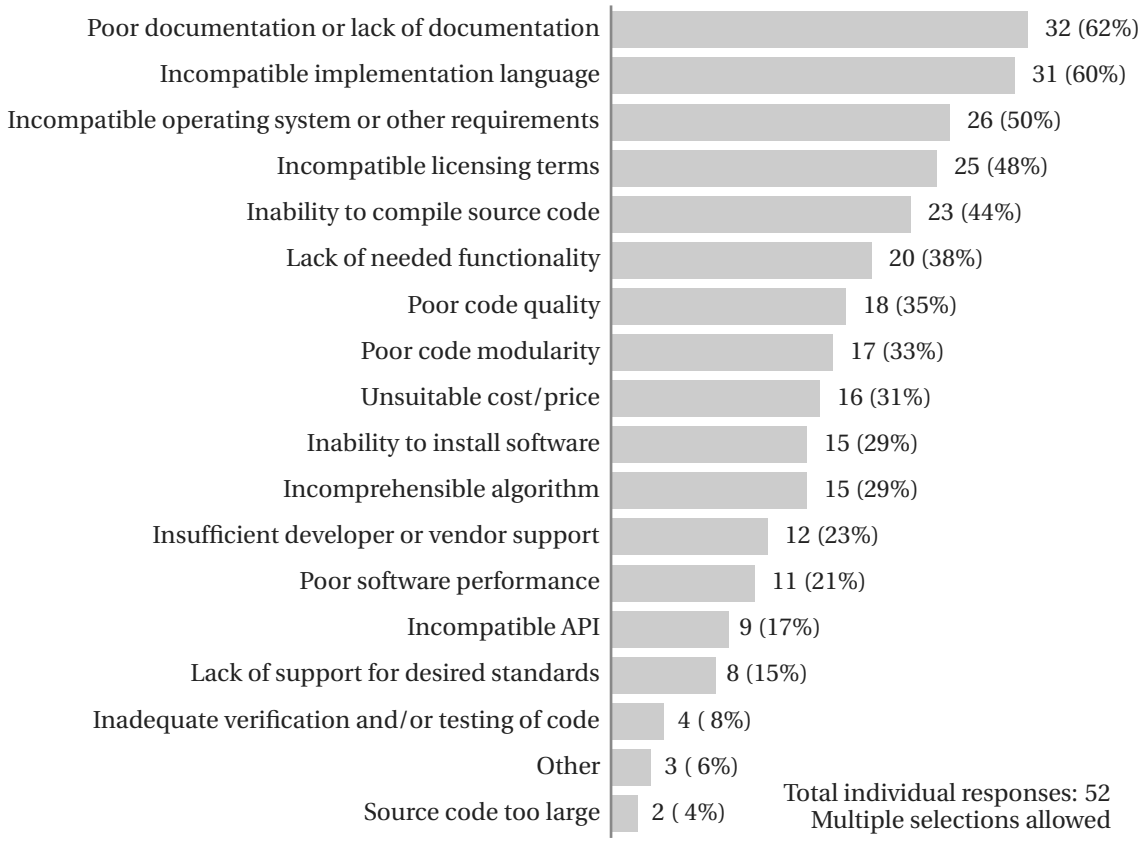

Figure 14: Responses to "If you searched and found source code in the past, what are some factors that may have prevented you from REUSING the source code you found?" The first 17 options were predefined; a free-text slot for "Other" was also provided. Answer choices were nonexclusive. A total of 52 survey participants answered this question.

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 15 on the next page summarizes the results. We separated the responses based on how individuals answered the yes/no question about being involved in software development (Section 3). The graph in Figure 15 is sorted by sum of responses across developers and nondevelopers for each answer category.

When viewed across both subgroups, 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 and programming language as characteristics they want to see indexed. In other respects, the general 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, as many

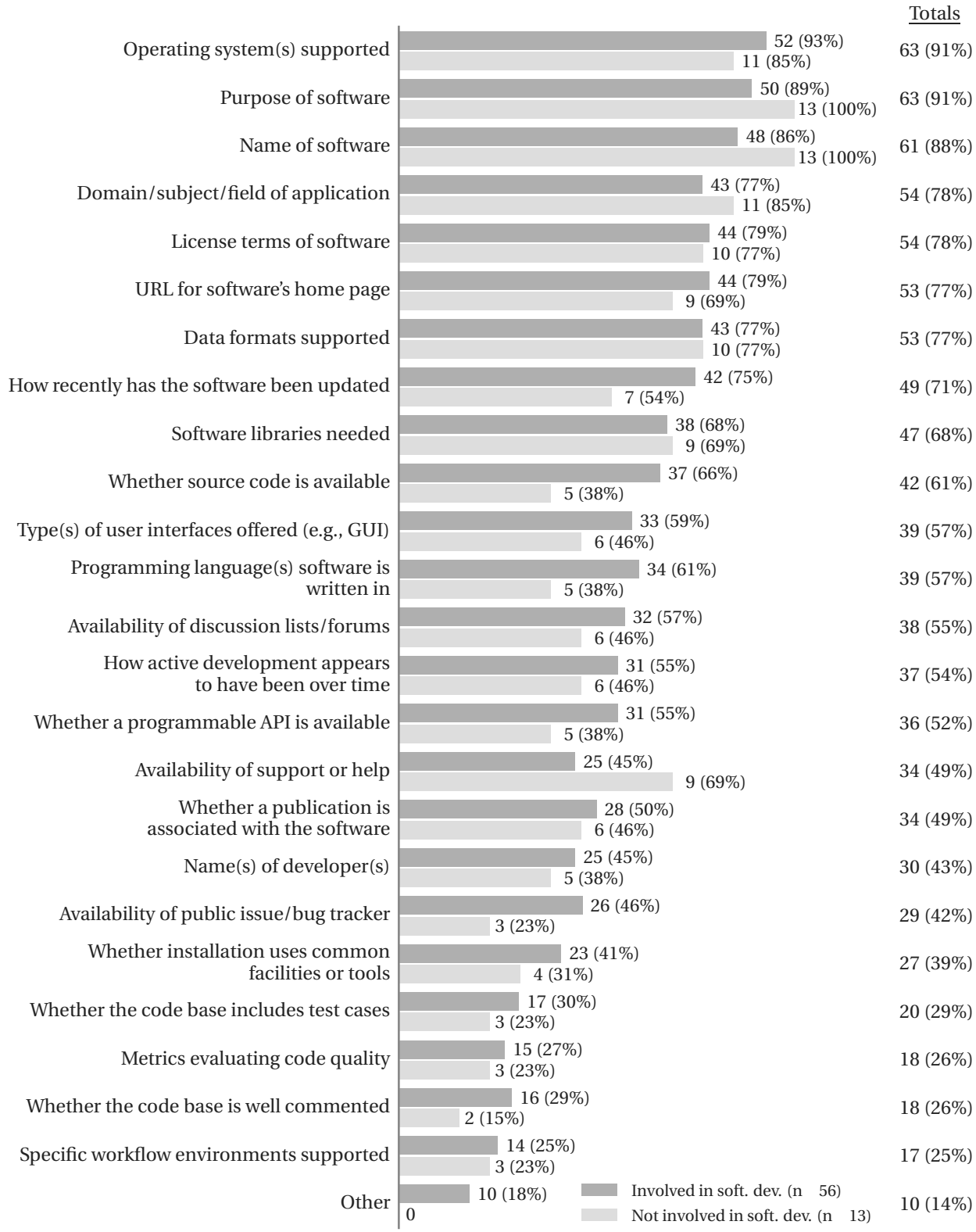

Figure 15: Responses to the question "Suppose that it were possible to create a public, searchable catalog or index of software, one that would record information about software of all kinds found anywhere. What kind of information would you find most useful to include for each entry in such a catalog 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 on page 5 (left). The graph is sorted by totals; e.g., "Name of software" was the third most selected choice across both developers and nondevelopers.

developers chose the availability of bug/issue trackers as often as they chose availability of support or help. A possible explanation for this is 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

6. Results: survey feedback

In the final question of the survey, we asked respondents to provide feedback on the survey itself. The 14 comments we received were largely positive, with one or two suggestions about the survey style. Many expressed the opinion that the amount of software available in the world made the construction of a software catalog or index a difficult task but, importantly, that it would a worthwhile endeavor. A few also commented on the broader issue of searching for software, saying that "although [they] do this quite often, [they] never really considered how or why code search succeed or not", and the survey "caused a bit of thinking about issues I rarely consider." One response was more specific on this point: "I wasn't conscious of having explicitly searched for software for a long time. I had to think about it, but turns out I do, quite a lot—but indirectly. I don't search for software—I search for solutions. I search for information about a specific issue, using a fragment from a stack trace, a specific error code, or two or three key words to describe the issue... An indirect reference, such as a one-liner that says 'use xyz to check the status' is more likely to result in a follow up than a whole article written specifically about a product."

7. Related work

As part of this project, we performed a literature survey to seek out research on topics relevant to those in our survey. We especially sought out other work that may have surveyed how people find or discover software. We discuss our findings in this section.

7.1. Surveys examining how software users find ready-to-run software

Though many surveys have examined software developers and search characteristics in the context of software *code* reuse, extremely few have examined how users—whether they are developers or not—go about locating *ready-to-run* software. Our research uncovered only three reports of surveys that were not focused specifically on a software development context [35, 41, 48].

Joppa et al. [41] surveyed 596 scientists working in a single domain (biological species distribution modeling), and asked them what software they used and why they chose that particular software. The reasons given by the respondents provide some insight into how the scientists found the software they used, thus addressing indirectly the same topic as one of our survey questions (see Section 4.1.1 on page 7). In order of most popular to least, 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%)
- "A reviewer suggested I use it" (1%)

It is interesting to note that none of the responses in Joppa et al.'s [41] survey explicitly mentioned searching the Internet via (e.g.) a Web search engine, 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. [35] summarized interviews of 15 students and faculty working in bioinformatics. They found that four factors influenced the selection of scientific software: (1) suggestions from mentors or senior members of an institution; (2) mentor involvement in the tool's development, in cases where mentors are also developers; (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. [47, 48] conducted a large survey in 2014 about the use of science gateways by members of scientific communities. Their nearly 5000 respondents consisted largely of NSF-funded principal investigators, as well as members of scientific computing centers and others who expressed interest in science gateways. Several of their questions and results are highly relevant to the topics of our own survey:

- They asked participants to indicate domains of expertise, and permitted multiple selections. 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. Their top three fields are thus very similar to those chosen by our survey respondents (see Figure 1 on page 4), though the proportions of the three are different.

- They asked how people learn about and choose gateways—a question closely related to our own survey's question about finding software (see Figure 6 on page 7). 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.

- They asked participants about the types of Web-based resources that were important to their work from the perspective of researchers and/or educators. The form included a matrix of ten predefined options and five-term rating scale ("Very important", "Somewhat important", etc.). The five highest rated resources where: "Data collections" (75% indicated somewhat or very important), "Data

analysis tools, including visualization and mining" (72%), "Computational tools" (72%), "Tools for rapidly publishing and/or finding articles and data" (69%), and "Educational tools" (67%).

- In another question, Lawrence et al. [48] 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 two of our survey's questions (see Figure 7 on page 9 and Figure 15 on page 18), 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%)

The last outcome summarized above from Lawrence et al.'s survey [48] is surprisingly different from the results of a similar question in our survey. Comparing the results above to our Figure 7 on page 9, neither the relative ordering of the chosen criteria, nor the absolute positions, are consistent between the two sets of survey responses. The differences are puzzling. 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. 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. In an effort to assess whether this is a possible cause, we reanalyzed our results by ranking the responses based on the sum of the number of times "Essential", "Usually of above-average importance", and "Average importance" were selected for each criterion. The results are shown in Figure 16 on the following page. Now the quality of documentation is sixth highest in rank, and quality of the software is fourth highest, which begins to approach the high ranks these two criteria had in Lawrence et al.'s survey. However, other features remain quite different in rankings between the survey results.

Other factors therefore must be responsible for the differences in results between our survey and that of Lawrence et al. [48]. 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. This may influence the criteria people use to select between software options. Second, Lawrence et al.'s survey had far more respondents than we did. It is possible that our results for this question would be different if we had a larger sample.

7.2. Surveys examining how developers reuse software

Most studies of how users find software have done so in the context of software development and the reuse of software 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

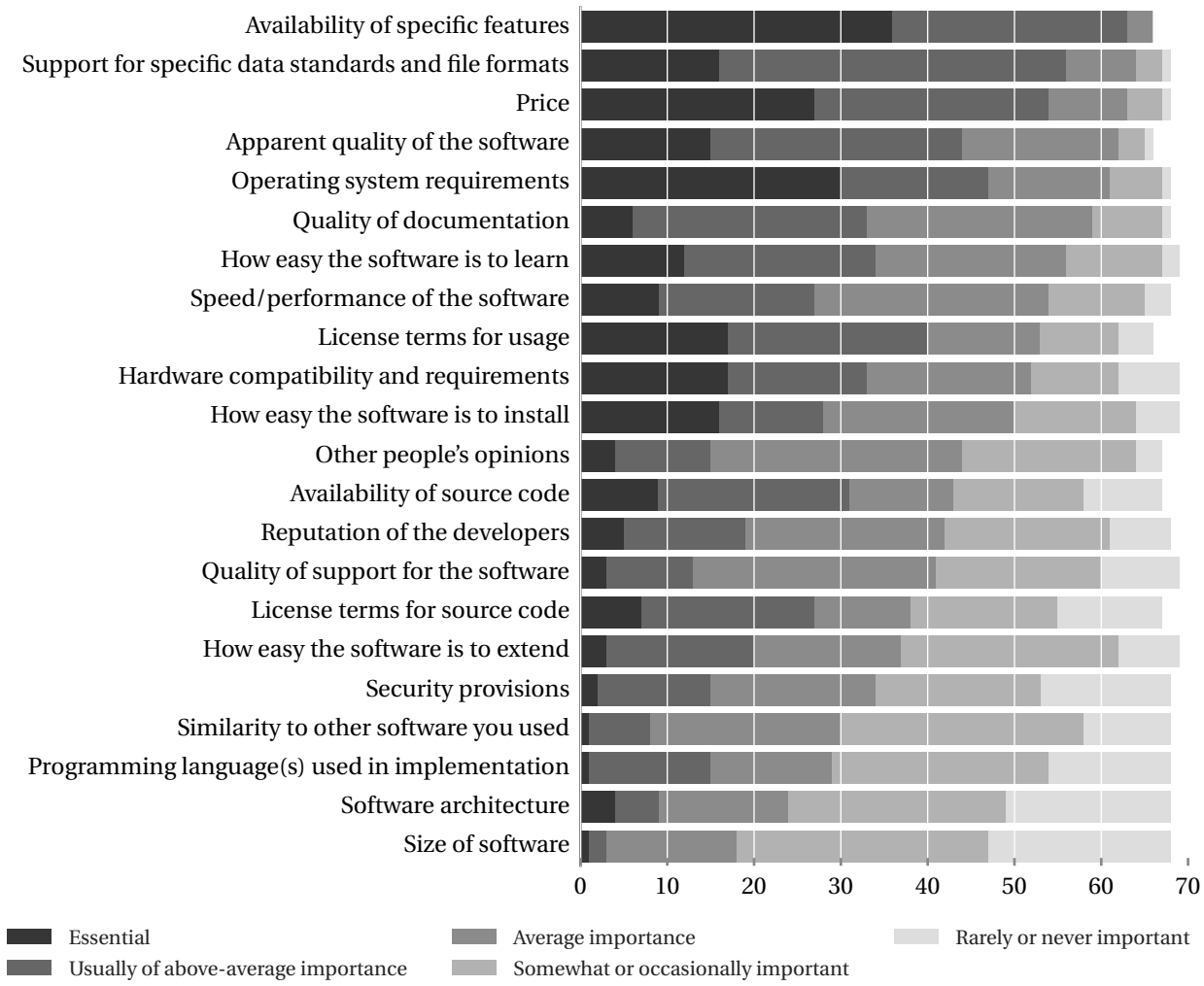

Figure 16: Reanalysis of the responses to the question of Figure 7 on page 9. In this bar graph, we sorted the possible criteria by the sum of the number of times the options "Essential", "Usually of above-average importance" and "Average importance" were selected for each one.

programming contexts, many studies examined the reuse of other kinds of artifacts such as documentation, specifications, architectural patterns, and more. For the purposes of this review, we discuss general studies about code reuse in this section, and leave those specifically about code search to Section 7.3.

One of the earliest surveys of software developers in the context of software reuse was published by Frakes and Fox [21]. In their 1991–1992 survey of 28 U.S. organizations, they received responses to 16 questions from 113 people about questions ranging from programming language and tool preferences to the impact of legal issues. Several results from this survey are relevant to the topics of our own survey:

- They found that reuse is more common in some industries than others. The subjects in Frakes and Fox's study [21] mostly came from high-technology sectors such as the software industry, aerospace and telecommunications. The authors found significant differences in the reuse of artifacts between different industries, as well as the types of artifacts that were reused. The telecommunications industry had the highest level of reuse and had the aerospace the lowest.

- They did not find that having a repository improved reuse. Frakes and Fox reported that "organizations with a repository have median code reuse levels 10 percent higher than organizations that do not have reuse repositories, but this difference is not statistically significant at the 0.05 level."

- The authors also examined whether company, division or project sizes were predictive of systematic reuse in an organization. They found no significant correlation between reuse levels and sizes.
- On the topic of whether participants prefer to develop their own software versus reusing someone else's, Frakes and Fox found that most of their respondents (72%) did not have a "not invented here" mentality—most developers prefer to reuse when possible.
- Several of the questions concerned factors that may or may not affect reuse behavior. We can summarize their results as follows: CASE (Computer Aided Software Engineering) tools do not promote reuse; education about reuse practices improves the level of reuse in an organization; reuse is higher in organizations having a process that promotes reuse; recognition for practicing reuse does not increase reuse by individuals, but monetary rewards do; and satisfaction with the quality of reusable artifacts did not affect reuse levels among the participants (but this was because the level of quality they encountered had been adequate).

Samadi et al. [64] reported preliminary findings from a 2004 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, and asked about people's recent reuse experiences and community needs. Several results from the study are pertinent to our work:

- 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), (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".
- As far as 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". Further, 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 further report that (a) availability of support, (b) standards compliance, and (c) testing/certification, were "not ranked as particularly important".
- On the topic of barriers to reuse, two common types of barriers emerged: (1) if available software did not meet a person's specific requirements, and (2) if a given software artifact was "difficult to understand or poorly documented".

The last result above from Samadi et al. [64] matches the results of our survey. For our question about what factors hindered people from finding software (Section 4.2.3), the most popular reason was finding a match to one's needs, and for our question about what hindered people's ability to reuse source code found (Section 4.2.4), quality of documentation was the most popular reason.

The study was reprised in 2005 with a wider audience that included members of academia. The results of 100 responses they received are summarized by Marshall et al. [52]. According to Marshall et al., the larger 2005 survey produced essentially similar results to the preliminary 2004 survey. They noted the following:

- The primary reason given by people for not reusing software from outside of their group was that "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, the results reported by Marshall et al. [52] seem to be inconsistent. The authors noted that "Web searches were of average importance while serendipity and reuse catalogs or repositories were rated the lowest"; however, in another section of the survey dealing with how to increase reuse within the Earth science community, one of the top three factors was "having an Earth science catalog/repository for reusable artifacts." In other words, catalogs were rated low in one part of the survey but high in another part. Despite this, among their conclusions, 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-centered study, Orrego and Mundy [59] examined software reuse in the context of flight control systems at NASA's Independent Verification & Validation (IV&V) Facility. They studied 63 projects using interviews, surveys and case studies. In interviews with 15 people, they found that black-box reuse and other types of reuse did occur, and the degree to which a given project reused software ranged from 0% to 80%. 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 components to be reused. Unfortunately, the Orrego and Mundy did not report the specific approaches attempted by people to locate software.

Singer et al. [75] 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 increase their awareness of people, trends and practices by subscribing to Twitter accounts by (a) individual developers as well as software project news channels relevant to their work, (b) news services or news curators, (c) "thought leaders" or experts in different subject areas.

- Developers extend their knowledge of software (including new software tools and components) by asking and answering questions, participating 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. [6] describe a study of reuse practices by developers at Google. Despite the nature of Google as one of the preeminent software organizations today, it is surprising that there is no centrally-controlled mandate about reuse of software. The question of what to reuse, and how, is left to engineers and managers. (There are core libraries and software components that get reused company-wide; these are under the care of dedicated teams, but the decision of what to use for a given task or application is evidently up to individuals and product teams.) Several of the questions in Bauer et al.'s survey [6] 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%). It is worth noting that Google has a centralized code repository where most of the code is available for all projects; this repository system features a centralized code search facility.

- 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%).

A final study relevant to this topic is the work of Lawrence et al. [47, 48], discussed in the previous section. An additional question in their survey is relevant in the context of software search and reuse by developers. 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%).

7.3. Surveys examining code search by developers

It has been long known from studies of developers at work [e.g., 74] that search is always a common activity. A number of studies have also examined more specifically how developers use search to discover software. It is important to keep mind, however, that while they are relevant to the general question of how users find software, such studies presuppose an answer: the users are *performing search* on a computer, and not (say) reading papers or asking colleagues for recommendations. Thus, the studies have a narrower scope than our survey, both in terms of the populations studied and in terms of approaches to finding software. On the other hand, they can delve more deeply into the details of how, where, when, and why the subjects searched for software.

Umarji et al. [84, 85] 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. In the 2008 paper and the 2013 book chapter, they focused on one of the survey questions asking people to describe 1–2 scenarios in which they looked for source code on the Internet. (A similar but earlier study by Sim et al. [73] predated the common use of Internet search for code; it has less relevance to the present work, so we do not report on it here.) Several facets of the Umarji et al. study are especially relevant to our own survey:

- With respect to why developers searched, out of a total of 51 searches described, Umarji et al. [85] found that the largest fraction were concerned with finding either (a) reusable code (67% of the 51 results), (b) reference examples (33%), or debugging activities (10%). Within the reuse category (a), the authors identified four themes: (1) search for code fragments; (2) search for subsystems that implemented reusable data structures, algorithms, or other elements that could be incorporated into an implementation; (3) search for packages or API libraries; and (4) search for stand-alone tools or systems (in the words of a participant, "big piece of code that does more or less what I want"). Within the category of reference examples (b), Umarji at al. also identified four (different) themes: (1) search for code fragments that illustrate syntax; (2) search for implementations of data structure, algorithm or widgets in order to verify a programmer's own approach or use as a basis for reimplementation; (3) search for examples of how to use a library; and (4) search for similar software to generate new ideas. Within the category of debugging (c), developers searched for solutions to software defects ("patches") or explanations for the cause of an error.

- Umarji and Sim [84] report that common starting points for searches were: (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 [76] (49%), (4) references from peers (43%), (4) mailing lists (23%), and (5) code-specific search engines such as Google Code Search [27] (16%).

- With respect to the selection criteria used by developers to choose a solution, Umarji and Sim [84] 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%).

Gallardo-Valencia and Sim [23] examined the Web search behaviors of 25 developers working at a software company that develops transactional software for banks and other organizations. The authors used several methods to gather their data: they asked developers to self-report their activities using record sheets, they performed interviews, and they performed observation of developer activities during their work days. They reported that 8% of the Web searches performed by developers during the study period were to download libraries or other tools, and another 5% of the Web searches were to collect information to help judge the suitability of software components to be used in implementations. The rest of the searches concerned finding information about how to do specific tasks or write specific kinds of programs, and debugging problems involving errors in software.

Sim et al. [69] performed a pair of surveys as part of an effort to understand developers' approaches to code search. One survey was exploratory and used to inform a second, quantitative survey. They report only a small part of the survey results, specifically concerning the nature of the searches performed. For purposes of the current review, the quantitative survey results are more relevant. On the topic of what subjects searched for, 92% of respondents in their survey answered they had searched for code snippets on past occasions, and 69% said they had searched for software components. In terms of motivations for why they conducted searches, 96% of participants said they had sought reference examples on past occasions, and 35% said that they had searched for code to reuse as-is.

Sadowski et al. [63] surveyed and analyzed search behaviors of 40 software developers at Google, Inc. To implement the survey, they developed a Web browser extension that directed developers to a survey system whenever the developers accessed an internal code search system at the company, and asked the survey participants to install this browser extension when they worked. The survey system asked developers four multiple-choice questions before they started a search. Sadowski et al. also analyzed search logs, but their survey is most pertinent to our efforts. Two questions are directly related to questions we also asked in our own survey:

- They investigated why developers searched and what questions they tried to answer with their code search. The most common theme (33.5% of the survey responses) dealt with getting specific information about an API library or examples of its use.

- Sadowski et al. also investigated the contexts in which code search is used. The most common situation (39% of survey responses) was performing searches while working on a code change during development. In this context, almost half of the developers (46%) used code search to understand how code worked.

7.4. Other studies

In addition to the related surveys reported here, there are other works not reviewed here that have applied other methods to examining developer behavior. The two main classes of non-survey techniques have been the analysis of search engine logs [3, 4, 10, 11, 25, 39, 49, 82, 89], and observational studies (often coupled with interviews), either in institutional settings or in laboratory environments [5, 10, 19, 24, 55, 61, 68, 70–72]. Due to the different techniques and intentions behind these efforts, the results are difficult to compare directly to our survey. Nevertheless, in future work we may seek to analyze them more systematically, with the goal of extracting information about how the subjects proceeded to find software and what features subjects found most useful when selecting between software options.

8. Conclusions

Before the advent of the World Wide Web, before even the advent of the current Internet, it was paradoxically easier to find software—there was less of it, and there were simply fewer places to look. Initially, bulletin boards and archive sites using FTP made software available for copying by anonymous users over telephone networks; the Usenet culture [20] of the 1980's encouraged widespread sharing and even devoted a newsgroup (*comp.sources*) to the exchange of software source code. 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.

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 higher than some other studies have reported (e.g., Hannay et al. [30] found scientists spent 40% of their time using scientific software), but this includes any software use, not only 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, the top five approaches overall are: (i) search the Web with general-purpose search engines, (ii) ask colleagues, (iii) look in the scientific literature, (iv) ask on social help sites such as Stack Overflow, and (v) use whatever is determined by the guidelines or work practices of the group or organization where one is working. However, the popularity of these approaches differed between software developers and nondevelopers: more nondevelopers chose asking colleagues than searching the Web, and far more developers than nondevelopers indicated they used social help sites such as Stack Overflow.

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) operating system requirements, and (v) apparent quality of the software.

3. The top five approaches used by developers to search for source code are almost identical to those used 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. Somewhat surprisingly, developers were more likely to search programming-oriented sites such as Stack Overflow when looking for *ready-to-run* software than when looking for source code.

4. The top five reasons developers search for source code are: (i) to find examples of how specific APIs are used, (ii) to find examples of how to implement something, (iii) to reuse code as-is, (iv) to remember syntactic details or infrequently used functionality, and (v) to learn unfamiliar concepts.

5. The top five reasons developers are unable to reuse the code they find are: (i) poor documentation, (ii) incompatible implementation language, (iii) incompatible operating system or other requirement, (iv) incompatible licensing terms, and (v) inability to compile source code.

6. Finally, we also asked people to indicate the information they would like to see in a software catalog. 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.

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 15 on page 18).

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 3), 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 Section 5, which included workflow environments as an answer choice. Based on the responses reported in Figure 15 on page 18, 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., 14, 15, 46]; 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., 6, 42, 48, 78, 93], the values often have been higher. For example, Sojer [78] reported 9.7% and Lawrence et al. [48] 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 on page 5, respondents had a mean of 20 years of experience, and in Figure 12 on page 14, 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.

In conclusion, 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.

9. Acknowledgments

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

Appendix A. The Survey

We implemented the survey using Google Forms [29], a free online system for creating interactive, Web-based forms. The following pages show screen captures of the survey as it appeared to users. Not shown is the switching logic that allowed some parts of the survey to be showed only if users responded in certain ways to some of the questions. The following are the rules implemented in the switching logic:

Rule for Question 6: If the user answers "I never get to choose the software I use", the next page shown is the final page of the survey (containing questions 21 and 22).

Rule for Question 11: If the user answers "No", the next page shown is the final page of the survey.

Rule for Question 16: If the user answers "Never", the next page shown is the final page of the survey.

Software search and software catalogs survey

Your answers to this survey's questions will help us understand some of the approaches people use to find software today, and will help us plan the development of facilities that could help people find software more easily in the future.

This survey is voluntary and you are not required to fill it out. Your identity will be kept confidential, even when we publish the results of this survey. To proceed, however, we need your explicit consent allowing us to store and use your replies to this survey. Please read the document located at http://goo.gl/q6yN4U and in the first question below, indicate whether you give your consent. (Note: the document covers more cases than this survey – you can ignore parts dealing with interviews and requesting signatures.)

This survey has up to 5 pages. The number of pages varies based on answers given to the questions.

* Required

Voluntary consent *

By checking the box below, you indicate that (1) you have read the informed consent form document linked above and (2) you consent to us using your responses to this survey in both our research and our future publications.

- [ ] I give my consent

What is your name?

This information will be kept confidential. We ask for your name (1) because the answer to the previous question must be associated with each individual, and (2) so that we know how to refer to you if we communicate with you.

What is your email address?

This information will be kept confidential. We will not release your address or other identifying information publicly. We ask for it so that we can contact you if necessary.

What is your primary field of work?

If work equally in multiple fields or across field boundaries, you may select more than one option from this list.

- [ ] Biology and Biological Engineering
- [ ] Chemistry and Chemical Engineering
- [ ] Cognitive and Brain Sciences
- [ ] Computing and Mathematical Sciences
- [ ] Geological and Planetary Sciences
- [ ] Physical Sciences
- [ ] Social Sciences
- [ ] Other:

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?

For most people, all work done with a computer or computing device involves interacting with software, because browsers, editors, etc., are all software systems. However, not all work involves computer work. The purpose of this question is to try to understand how much you interact with software programs in your work.

In your work, how much freedom do you usually have to choose the software you use?

This refers to situations where you USE software, rather than when you write software. In other words, it is about the software applications and environments you use. The possibilities range from having no choice (perhaps because you are required to use pre-selected software) to having complete autonomy (perhaps because there are no policies or guidelines, or because part of your work is to make the choices for yourself or for others).

- I never get to choose the software I use
- Sometimes I get to choose the software I use
- Half the time, a situation or task requires using preselected software, and other times I get to choose
- More often than not, I choose the software I use
- I always choose the software I use

Continue »

20% completed

Software search and software catalogs survey

Questions for software users

When you need to find ready-to-run software for a particular task, how do you go about finding software?

Note that this is about using software rather than developing software, so searching for software here does not necessarily imply searching for source code. Please select all that apply.

- ☐ Use whatever is determined by my organization or work group's guidelines or practices
- ☐ Ask colleagues for opinions
- ☐ Ask or search mailing lists or discussion groups within your organization
- ☐ Ask or search public mailing lists or discussion groups
- ☐ Look in the scientific literature to find what authors use in similar contexts
- ☐ Ask or search social media (e.g., Twitter, Facebook, LinkedIn, etc.)
- ☐ Ask or search social help sites (e.g., StackOverflow, Quora, etc.)
- ☐ Search the web using general-purpose search systems (e.g., Google, Yahoo, Bing, DuckDuckGo)
- ☐ Search in topical software indexes/catalogs (e.g., ASCL.net, BioPortal, Alternative.to, etc.)
- ☐ Search in public software project repositories (SourceForge, GitHub, BitBucket, etc.)
- ☐ Other:

In general, how important are the following characteristics when you are searching for ready-to-run software for a task?

Please rank the following according to the importance of each to you. If the criteria change depending on the occasion, please indicate the criteria that represent what you most often use.

	Rarely or never important	Somewhat or occasionally important	Average importance	Usually of above-average importance	Essential
Availability of specific features	○	○	○	○	○
Availability of source code	○	○	○	○	○
Support for specific data standards and file formats	○	○	○	○	○
How easy the software is to learn	○	○	○	○	○
How easy the software is to extend	○	○	○	○	○
How easy the software is to install	○	○	○	○	○
Apparent quality of the software	○	○	○	○	○
Reputation of the developers	○	○	○	○	○

	Rarely or never important	Somewhat or occasionally important	Average importance	Usually of above-average importance	Essential
Quality of support for the software	○	○	○	○	○
Quality of documentation	○	○	○	○	○
Other people's opinions	○	○	○	○	○
Speed/performance of the software	○	○	○	○	○
Operating system requirements	○	○	○	○	○
Hardware compatibility and requirements	○	○	○	○	○
Similarity to other software you used	○	○	○	○	○
Programming language(s) used in implementation	○	○	○	○	○
Software architecture	○	○	○	○	○
Security provisions	○	○	○	○	○
Size of software	○	○	○	○	○
Price	○	○	○	○	○
License terms for usage	○	○	○	○	○
License terms for source code	○	○	○	○	○

(Optional) Please describe a past scenario when you looked for ready-to-run software.

Narratives like this help us understand the context of people's searches for software. Please address details such as: What were you trying to find? What information sources did you use to find the software? How did you formulate your questions or queries? What criteria did you use to decide on the best match? Were you successful in finding the software you were looking for? If unsuccessful, why?

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?

Please check all that apply. If you think of others, please write them in the "Other" slot.

- ☐ Name of software
- ☐ Domain/subject/field of application

- [ ] Purpose of software
- [ ] Name(s) of developer(s)
- [ ] Data formats supported
- [ ] License terms of software
- [ ] Operating system(s) supported
- [ ] Software libraries needed
- [ ] Programming language(s) software is written in
- [ ] How recently has the software been updated
- [ ] How active development appears to have been over time
- [ ] Availability of support or help
- [ ] Availability of public issue/bug tracker
- [ ] Availability of discussion lists/forums
- [ ] Whether the code base includes test cases
- [ ] Whether the code base is well commented
- [ ] Whether a programmable API is available
- [ ] Specific workflow environments supported
- [ ] Type(s) of user interfaces offered (e.g., GUI)
- [ ] Whether source code is available
- [ ] Whether installation uses common facilities or tools
- [ ] Whether a publication is associated with the software
- [ ] Metrics evaluating code quality
- [ ] URL for software's home page
- [ ] Other:

Are you involved in software development?

For the purposes of this question, it does not matter whether you do it alone or as part of a team or group.

- ○ Yes
- ○ No

« Back Continue »

40% completed

Software search and software catalogs survey

Questions for software developers

For how many years have you been developing software?

For the purposes of this question, please count all your lifelong software experiences, whether for your work or for personal projects.

In your current (or most recent) software development project, what is (or was) your primary responsibility?

If you have multiple roles on a project, what would be the primary one? (If it is hard to identify a single one, you can indicate more than one below.)

- [ ] Project management
- [ ] Requirements analysis
- [ ] Software architecture
- [ ] Software development
- [ ] Testing/quality assurance
- [ ] Technical writing
- [ ] Deployment
- [ ] Training
- [ ] Other:

What is the typical team size of projects that you are involved with?

Here are are interested in all efforts you have been involved with, whether open-source or proprietary, and whether related to your current work or not.

- ○ Small (1–5 people)
- ○ Medium (6–25 people)
- ○ Large (more than 25 peple)
- ○ Other:

Which programming and/or scripting language(s) have you had the most experience with?

Please select up to 3 languages which you have used the most.

- [ ] C
- [ ] C++
- [ ] C#
- [ ] Delphi/Pascal/Object Pascal
- [ ] F#
- [ ] Fortran
- [ ] IDL
- [ ] Java

- [ ] JavaScript
- [ ] Lisp
- [ ] Mathematica
- [ ] MATLAB/Octave
- [ ] Objective-C
- [ ] Perl
- [ ] PHP
- [ ] Python
- [ ] R
- [ ] Ruby
- [ ] SQL
- [ ] Shell scripting (Unix/Linux/Mac OS)
- [ ] Visual Basic
- [ ] Windows batch file scripting
- [ ] Other:

How often do you search online for software source code?

- ( ) Never
- ( ) Rarely – once every few months
- ( ) Once per month, on average
- ( ) Once per week, on average
- ( ) Once per day, on average
- ( ) Many times per day

« Back Continue »

60% completed

Software search and software catalogs survey

Questions for developers who search for source code

What are some reasons why you look for source code (when you do look)?

Please check all that apply.

- [ ] To reuse code as-is (either in whole or in part)
- [ ] To find code examples of how specific APIs are used
- [ ] To find examples of how to implement a particular algorithm, data structure or approach
- [ ] To find a more efficient approach to something you have already written
- [ ] To remember syntactic details or infrequently used functionality
- [ ] To discover new algorithms or data structures
- [ ] To learn unfamiliar concepts
- [ ] To confirm or refute that some given software has a defect/bug
- [ ] Other:

What are some approaches you have used to look for source code in the past?

Please select all that apply.

- [ ] Ask colleagues for suggestions
- [ ] Ask or search mailing lists or discussion groups within to your organization
- [ ] Ask or search public mailing lists or discussion groups
- [ ] Look in the scientific literature to find what authors use in similar contexts
- [ ] Ask or search social media (e.g., Twitter, Facebook, LinkedIn, etc.)
- [ ] Ask or search social help sites (e.g., StackOverflow, Quora, etc.)
- [ ] Search the web using specialized code search engines (e.g., OpenHUB, Google Code, Krugle, Snipplr, Smipple, etc.)
- [ ] Search the web using general-purpose search systems (e.g., Google, Yahoo, Bing, DuckDuckGo)
- [ ] Search in public software project repositories (SourceForge, GitHub, BitBucket, etc.)
- [ ] Search in specialized software indexes/catalogs (e.g., ASCL.net, SBML Software Guide, BioPortal, etc.)
- [ ] Search in my organization's code collection or repository (if any)
- [ ] Other:

What are some factors that have hindered your ability to FIND source code in the past?

Please check all that apply.

- [ ] Lack of time to do a proper search and/or evaluate the results
- [ ] Unable to locate any suitable or working software source code for my purposes
- [ ] Concerns about intellectual property issues
- [ ] Lack of trust in the options found
- [ ] My specific requirements were too unique
- [ ] Using 3rd-party source code is prevented by policies

- [ ] Other:

If you searched and found source code in the past, what are some factors that may have prevented you from REUSING the source code you found?

You may have searched for source code in the past but not ended up being able to use what you found. What were some of the reasons? Please check all that apply.

- [ ] Incompatible licensing terms
- [ ] Incompatible implementation language
- [ ] Incompatible API
- [ ] Incompatible operating system or other requirements
- [ ] Inability to compile source code
- [ ] Inability to install software
- [ ] Poor code quality
- [ ] Poor code modularity
- [ ] Poor documentation or lack of documentation
- [ ] Poor software performance
- [ ] Lack of support for desired standards
- [ ] Lack of needed functionality
- [ ] Incomprehensible algorithm
- [ ] Inadequate verification and/or testing of code
- [ ] Insufficient developer or vendor support
- [ ] Unsuitable cost/price
- [ ] Source code too large
- [ ] Other:

« Back Continue »

80% completed

Software search and software catalogs survey

* Required

Thank you!

Thank you very much for taking the time to complete this survey.

May we contact you again to ask other questions in the future? *

For the purposes of this project only, we may want to ask further questions or even interview some respondents. You are under no obligation to accept, and even if you respond "yes" here, you can decline in the future.

- ○ Yes
- ○ No

We welcome your opinion about this survey – please let us know what you thought of it.

If you have any feedback (for example, you liked it, or you thought of some additional points to make, or were confused by some questions, or hated some part of it, or thought the whole thing was stupid, or whatever), please don't hesitate to write it here. We really want to know, and you can be completely honest here. We want to know!

Contact

If you have any questions, feedback, or other communications, please don't hesitate to email us at the following addresses:

Michael Hucka – mhucka@caltech.edu
Matthew Graham – mjg@caltech.edu

« Back | Submit

100%: You made it.

Never submit passwords through Google Forms.

Appendix B. Invitation letters

This section includes samples of the messages used to advertise the survey to different mailing lists and social groups.

Sample Message 1

```
Hi,

A colleague and I have been funded by NSF to do a pilot project on
"cataloging software using a semantic-based approach for software
discovery and characterization". As part of this, we are conducting an
initial survey to understand how people search for software and we would
appreciate it if you would consider taking part. The survey will take
10-15 mins and is available here:

https://goo.gl/W0jDMJ

Cheers,
```

Sample Message 2

```
Dear colleagues,

As part of an NSF-funded project to prototype a general software
cataloguing system, we're running a survey to assess how people look for
software today (biology or otherwise).  The motivation for the effort is
simple: everyone has faced the problem of finding a suitable software tool
for a given task, and we'd like to learn more about what approaches you've
used in the past so that we can help develop better ways of finding
software in the future.  This will benefit both software users and
developers.

Here is a link to the electronic survey form:

https://goo.gl/W0jDMJ

We estimate it takes 10-15 minutes to fill out.

Best regards,
```

References

[1] Alice Allen and Judy Schmidt. Looking before leaping: Creating a software registry. *Journal of Open Research Software*, 3(1), November 2015. doi: 10.5334/jors.bv.

[2] Alice Allen, Peter Teuben, Robert J. Nemiroff, and Lior Shamir. Practices in code discoverability: Astrophysics Source Code Library. In P. Ballester, editor, *Astronomical Data Analysis Software and Systems XXI*, volume 461, page 627, September 2012.

[3] Sushil Bajracharya and Cristina Lopes. 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*, 2009.

[4] Sushil Krishna Bajracharya and Cristina Videira Lopes. Analyzing and mining a code search engine usage log. *Empirical Software Engineering*, 17(4-5):424–466, 2012.

[5] Rajiv D. Banker, Robert J. Kauffman, and Dani Zweig. Repository evaluation of software reuse. *IEEE Transactions on Software Engineering*, 19(4):379–389, 1993.

[6] Veronika Bauer, Jonas Eckhardt, Benedikt Hauptmann, and Manuel Klimek. An exploratory study on reuse at Google. In *Proceedings of the 1st International Workshop on Software Engineering Research and Industrial Practices (SER&IPs 2014)*, pages 14–23. ACM Press, 2014.

[7] Black Duck Software, Inc. Black Duck Open Hub. Available on the World Wide Web at https://www.openhub.net/, 2016.

[8] Sebastian Bönisch, Michael Brickenstein, Hagen Chrapary, Gert-Martin Greuel, and Wolfram Sperber. swMATH – a new information service for mathematical software. In Jacques Carette, David Aspinall, Christoph Lange, Petr Sojka, and Wolfgang Windsteiger, editors, *Intelligent Computer Mathematics (Held as Part of CICM 2013, Bath, UK, July 8–12, 2013)*, Lecture Notes in Artificial Intelligence, pages 369–373. Springer, 2013.

[9] Phillip E. Bourne. Foundations for discovery informatics. Available on the World Wide Web at http://www.slideshare.net/pebourne/foundations-for-discovery-informatics, January 2015.

[10] Joel Brandt, Philip J. Guo, Joel Lewenstein, Mira Dontcheva, and Scott R. Klemmer. 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, pages 1589–1598, New York, NY, USA, 2009. ACM. doi: 10.1145/1518701.1518944.

[11] Joel Brandt, Mira Dontcheva, Marcos Weskamp, and Scott R. Klemmer. Example-centric programming: integrating web search into the development environment. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 513–522, 2010.

[12] Shirley Browne, Jack Dongarra, Eric Grosse, and Tom Rowan. The Netlib mathematical software repository. *D-Lib Magazine*, September 1995.

[13] Nicola Cannata, Emanuela Merelli, and Russ B. Altman. Time to organize the bioinformatics resourceome. *PLoS Computational Biology*, 1(7):e76, 2005.

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

[15] Mick P. Couper and Peter V. Miller. Web survey methods: Introduction. *Public Opinion Quarterly*, 72 (5):831–835, December 2008. doi: 10.1093/poq/nfn066.

[16] Sharon M. Crook, Andrew P. Davison, and Hans E. Plesser. Learning from the past: Approaches for reproducibility in computational neuroscience. In *20 Years of Computational Neuroscience*, pages 73–102. Springer, January 2013. doi: 10.1007/978-1-4614-1424-7_4.

[17] Kevin Crowston, Hala Annabi, and James Howison. Defining open source software project success. In *Proceedings of the Twenty-Fourth International Conference on Information Systems (ICIS 2003)*. Association for Information Systems, 2003.

[18] Kevin Crowston, James Howison, and Hala Annabi. Information systems success in free and open source software development: Theory and measures. *Software Process: Improvement and Practice*, 11(2):123–148, 2006.

[19] Laura Dabbish, Colleen Stuart, Jason Tsay, and Jim Herbsleb. Social coding in GitHub: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*, pages 1277–1286, 2012.

[20] Sandra L. Emerson. Usenet: A bulletin board for Unix users. *Byte*, 8(10):221–236, October 1983.

[21] William B. Frakes and Christopher J. Fox. Sixteen questions about software reuse. *Communications of the ACM*, 38(6):75–87, 1995.

[22] Landon Fuller, Kevin Van Vechten, and Jordan Hubbard. The MacPorts project – home. Available on the World Wide Web at https://www.macports.org/, 2002.

[23] Rosalva E. Gallardo-Valencia and Susan Elliott Sim. 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*, pages 41–44, 2011.

[24] Rosalva E. Gallardo-Valencia and Susan Elliott Sim. Software problems that motivate web searches. In *Finding Source Code on the Web for Remix and Reuse*, chapter 13, pages 253–270. Springer, 2013.

[25] Xi Ge, David Shepherd, Kostadin Damcvski, and Emerson Murphy-Hill. How developers use multi-recommendation system in local code search. In *Visual Languages and Human-Centric Computing (VL/HCC), 2014 IEEE Symposium on*, pages 69–76, 2014.

[26] Padraig Gleeson. Current application support for NeuroML. Available on the World Wide Web at https://www.neuroml.org/tool_support, 2016.

[27] Google, Inc. 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, 2006.

[28] Google, Inc. Google Shortener. Available on the World Wide Web at http://goo.gl, 2015.

[29] Google, Inc. Google Forms. Available on the World Wide Web at https://www.google.com/forms/about/, 2015. Accessed 2015-09-01.

[30] Jo Erskine Hannay, Carolyn MacLeod, Janice Singer, Hans Petter Langtangen, Dietmar Pfahl, and Greg Wilson. How do scientists develop and use scientific software? In *Proceedings of the 2009 ICSE workshop on Software Engineering for Computational Science and Engineering*, SECSE '09, pages 1–8, Washington, DC, USA, 2009. doi: 10.1109/SECSE.2009.5069155.

[31] Chris Hempel, Maytal Dahan, Carrie Arnold, Rion Dooley, Matthew Hanlon, Susan Lindsey, Stephen Mock, David Montoya, Alex Rocha, Manuel Rojas, and Walter Scarborough. XSEDE user portal: Software search. Available on the World Wide Web at https://www.xsede.org/software, 2016.

[32] Simon Hettrick. 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/2014-12-04-its-impossible-conduct-research-without-software-say-7-out-10-uk\-researchers. Archived at http://perma.cc/4M65-3WUP, December 2014.

[33] James Howison and Julia Bullard. 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*, May 2015. doi: 10.1002/asi.23538.

[34] James Howison, Ewa Deelman, Michael J. McLennan, Rafael Ferreira da Silva, and James D. Herbsleb. Understanding the scientific software ecosystem and its impact: Current and future measures. *Research Evaluation*, 24(4):454–470, July 2015. doi: 10.1093/reseval/rvv014.

[35] Xing Huang, Tun Lu, Xianghua Ding, Tiejiang Liu, and Ning Gu. 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*, pages 172–177, 2013.

[36] Michael Hucka, Frank T. Bergmann, and Bruce E. Shapiro. 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, 2016.

[37] John D. Hunter. Matplotlib: A 2D graphics environment. *Computing In Science & Engineering*, 9(3): 90–95, 2007.

[38] Darrel C. Ince, Leslie Hatton, and John Graham-Cumming. The case for open computer programs. *Nature*, 482(7386):485, February 2012. doi: 10.1038/nature10836.

[39] Bernard J. Jansen and Amanda Spink. How are we searching the world wide web? a comparison of nine search engine transaction logs. *Information Processing & Management*, 42(1):248–263, January 2006. doi: 10.1016/j.ipm.2004.10.007.

[40] Ola Johansson and Markus Olausson. 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, 2016.

[41] Lucas N. Joppa, Greg McInerny, Richard Harper, Lara Salido, Kenji Takeda, Kenton O'Hara, David Gavaghan, and Stephen Emmott. Troubling trends in scientific software use. *Science*, 340(6134): 814–815, May 2013. doi: 10.1126/science.1231535.

[42] Eirini Kalliamvakou, Georgios Gousios, Kelly Blincoe, Leif Singer, Daniel M. German, and Daniela Damian. The promises and perils of mining GitHub. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 92–101, 2014.

[43] Daniel S. Katz. 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, February 2015.

[44] Daniel S. Katz and Rajiv Ramnath. Looking at software sustainability and productivity challenges from NSF. *Computing Resources Repository*, abs/1508.03348, August 2015. URL http://arxiv.org/abs/1508.03348.

[45] Daniel S. Katz, Sou-Cheng T. Choi, Kyle E. Niemeyer, James Hetherington, Frank Löffler, Dan Gunter, Ray Idaszak, Steven R. Brandt, Mark A. Miller, Sandra Gesing, Nick D. Jones, Nic Weber, Suresh Marru, Gabrielle Allen, Birgit Penzenstadler, Colin C. Venters, Ethan Davis, Lorraine Hwang, Ilian Todorov, Abani Patra, and Miguel de Val-Borro. Report on the third workshop on sustainable software for

science: Practice and experiences (WSSSPE3). *Computing Research Repository*, abs/1602.02296 (arXiv:1602.02296), February 2016.

[46] Barbara A. Kitchenham and Shari L. Pfleeger. Personal opinion surveys. In Forrest Shull, Janice Singer, and Dag I.K. Sjøberg, editors, *Guide to advanced empirical software engineering*, chapter 3, pages 63–92. Springer-Verlag, 2008.

[47] Katherine A. Lawrence, Nancy Wilkins-Diehr, Julie A. Wernert, Marlon Pierce, Michael Zentner, and Suresh Marru. Who cares about science gateways?: A large-scale survey of community use and needs. In *Proceedings of the 9th Gateway Computing Environments Workshop*, pages 1–4, Piscataway, NJ, USA, 2014. IEEE Press. doi: 10.1109/GCE.2014.11.

[48] Katherine A. Lawrence, Michael Zentner, Nancy Wilkins-Diehr, Julie A. Wernert, Marlon Pierce, Suresh Marru, and Scott Michael. 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, May 2015. doi: 10.1002/cpe.3526.

[49] Yan Li, Lu Zhang, Bing Xie, and Jiasu Sun. Refining component description by leveraging user query logs. *Journal of Systems and Software*, 82(5):751 – 758, 2009. doi: 10.1016/j.jss.2008.10.027.

[50] Erik Linstead, Sushil Bajracharya, Trung Ngo, Paul Rigor, Critina Lopes, and Pierre Baldi. Sourcerer: mining and searching internet-scale software repositories. *Data Mining and Knowledge Discovery*, 18(2):300–336, 2009.

[51] Mario. 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, 2016.

[52] James J. Marshall, Stephen W. Olding, Robert E. Wolfe, and Victor E. Delnore. Software reuse within the earth science community. In *Geoscience and Remote Sensing Symposium, 2006. IGARSS 2006. IEEE International Conference on*, pages 2880–2883, 2006.

[53] Eduardo Mena, Arantza Illarramendi, Jose A. Royo, and Alfredo GoñI. A software retrieval service based on adaptive knowledge-driven agents for wireless environments. *ACM Transactions on Autonomous and Adaptive Systems*, 1(1):67–90, September 2006. doi: 10.1145/1152934.1152938.

[54] A. Morin, J. Urban, P.D. Adams, I. Foster, A. Sali, D. Baker, and P. Sliz. Shining light into black boxes. *Science*, 336:159–160, April 2012.

[55] Emerson Murphy-Hill, Da Young Lee, Gail C. Murphy, and Joanna McGrenere. How do users discover new tools in software development and beyond? *Computer Supported Cooperative Work (CSCW)*, 24 (5):389–422, July 2015. doi: 10.1007/s10606-015-9230-9.

[56] National Aeronautics and Space Administration. 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, 2016.

[57] Kyle E. Niemeyer, Arfon M. Smith, and Daniel S. Katz. The challenge and promise of software citation for credit, identification, discovery, and reuse. *Computing Research Repository*, abs/1601.04734 (arXiv:1601.04734), January 2016.

[58] Natalya F. Noy, Nigam H. Shah, Patricia L. Whetzel, Benjamin Dai, Michael Dorf, Nicholas Griffith, Clement Jonquet, Daniel L. Rubin, Margaret-Anne Storey, Christopher G. Chute, and Mark A. Musen. BioPortal: ontologies and integrated data resources at the click of a mouse. *Nucleic Acids Research*, 37:W170–3, July 2009. doi: 10.1093/nar/gkp440.

[59] Andres S. Orrego and Gregory E. Mundy. A study of software reuse in NASA legacy systems. *Innovations in Systems and Software Engineering*, 3(3):167–180, July 2007. doi: 10.1007/s11334-007-0027-y.

[60] Fernando Perez, Brian E. Granger, and John D. Hunter. Python: an ecosystem for scientific computing. *Computing in Science & Engineering*, 13(2):13–21, 2011.

[61] Amnart Pohthong and David Budgen. Reuse strategies in software development: an empirical study. *Information and Software Technology*, 43(9):561–575, 2001.

[62] Timothee Poisot. Best publishing practices to improve user confidence in scientific software. *Ideas in Ecology and Evolution*, 8, 2015. doi: 10.4033/iee.2015.8.8.f.

[63] Caitlin Sadowski, Kathryn T. Stolee, and Sebastian Elbaum. How developers search for code: a case study. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 191–201, 2015.

[64] Shahin Samadi, Nadine Almaeh, Robert Wolfe, Steve Olding, and David Isaac. Strategies for enabling software reuse within the earth science community. In *Proceedings of the 2004 IEEE International Geoscience and Remote Sensing Symposium (IGARSS'04)*, volume 3, pages 2196–2199, 2004.

[65] National Research Council Committee on Responsibilities of Authorship in the Biological Sciences. *Sharing Publication-related Data and Materials: Responsibilities of Authorship in the Life Sciences*. National Academies Press, 2003.

[66] Ravi Sen, Siddhartha S. Singh, and Sharad Borle. Open source software success: Measures and analysis. *Decision Support Systems*, 52(2):364–372, January 2012. doi: 10.1016/j.dss.2011.09.003.

[67] Wade Shen. 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, November 2015.

[68] Karma Sherif and Ajay Vinze. Barriers to adoption of software reuse: a qualitative study. *Information & Management*, 41(2):159–175, 2003.

[69] Susan Elliot Sim, Rosalva Gallardo-Valencia, Kavita Philip, Medha Umarji, Megha Agarwala, Cristina V. Lopes, and Sukanya Ratanotayanon. Software reuse through methodical component reuse and amethodical snippet remixing. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*, pages 1361–1370, 2012.

[70] Susan Elliott Sim and Thomas A. Alspaugh. Getting the whole story: an experience report on analyzing data elicited using the war stories procedure. *Empirical Software Engineering*, 16(4): 460–486, 2011.

[71] Susan Elliott Sim, Medha Umarji, Sukanya Ratanotayanon, and Cristina V. Lopes. How well do search engines support code retrieval on the web? *ACM Transactions on Software Engineering and Methodology*, 21(1):1–25, December 2011. doi: 10.1145/2063239.2063243.

[72] Susan Elliott Sim, Megha Agarwala, and Medha Umarji. 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*, chapter 4, pages 53–77. Springer, 2013.

[73] Susan Elliott G. Sim, Charles L.A. Clarke, and Richard C. Holt. Archetypal source code searches: A survey of software developers and maintainers. In *Proceedings of the Sixth International Workshop on Program Comprehension*, pages 180–187. IEEE, 1998.

[74] Janice Singer, Timothy Lethbridge, Norman Vinson, and Nicolas Anquetil. An examination of software engineering work practices. In *Proceedings of the 1997 Conference of the Centre for Advanced Studies on Collaborative Research (CASCON '97)*, pages 174–188, 1997.

[75] Leif Singer, Fernando Figueira Filho, and Margaret-Anne . A. Storey. 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, pages 211–221, New York, NY, USA, 2014. ACM. doi: 10.1145/2568225.2568305.

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

[77] SlashDot Media. SourceForge directory. Available on the World Wide Web at https://sourceforge.net/directory, 2016.

[78] Manuel Sojer. *Reusing Open Source Code*. PhD thesis, Technische Universität München, 2010.

[79] Stack Exchange Inc. 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, March 2016.

[80] Craig A. Stewart, Julie Wernert, Eric A. Wernert, William K. Barnett, and Von Welch. Initial findings from a study of best practices and models for cyberinfrastructure software sustainability. *Computing Research Repository*, abs/1309.1817, 2013.

[81] Chandrasekar Subramaniam, Ravi Sen, and Matthew L. Nelson. Determinants of open source software project success: A longitudinal study. *Decision Support Systems*, 46(2):576–585, January 2009. doi: 10.1016/j.dss.2008.10.005.

[82] Jaime Teevan, Christine Alvarado, Mark S. Ackerman, and David R. Karger. 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, pages 415–422, New York, NY, USA, 2004. ACM. doi: 10.1145/985692.985745.

[83] Sang-Yong Tom Lee, Hee-Woong Kim, and Sumeet Gupta. Measuring open source software success. *Omega: The International Journal of Management Science*, 37(2):426–438, 2009.

[84] Medha Umarji and Susan Elliot Sim. Archetypal internet-scale source code searching. In Susan Elliot Sim and Rosalva E. Gallardo-Valencia, editors, *Finding Source Code on the Web for Remix and Reuse*, chapter 3. Springer Verlag, 2013.

[85] Medha Umarji, Susan Elliott Sim, and Crista Lopes. Archetypal internet-scale source code searching. In Barbara Russo, Ernesto Damiani, Scott Hissam, Björn Lundell, and Giancarlo Succi, editors, *Open Source Development, Communities and Quality*, IFIP International Federation for Information Processing, pages 257–263. Springer, 2008.

[86] Stefan Van Der Walt, S. Chris Colbert, and Gael Varoquaux. The NumPy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011.

[87] Guido van Rossum and Jeke de Boer. Interactively testing remote servers using the Python programming language. *CWI Quarterly*, 4(4):283–303, December 1991.

[88] Julia Varnell-Sarjeant, Anneliese Amschler Andrews, Joe Lucente, and Andreas Stefik. Comparing development approaches and reuse strategies: An empirical evaluation of developer views from the aerospace industry. *Information and Software Technology*, 61:71–92, May 2015. doi: 10.1016/j.infsof.2015.01.002.

[89] Michael Völske, Pavel Braslavski, Matthias Hagen, Galina Lezina, and Benno Stein. What users ask a search engine: Analyzing one billion russian question queries. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 1571–1580, 2015.

[90] Owen White, Asif Dhar, Vivien Bonazzi, Jennifer Couch, and Chris Wellington. 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, October 2014.

[91] Gregory V. Wilson. Where's the real bottleneck in scientific computing? *American Scientist*, 94(1):5, 2006. doi: computational.

[92] Jonathan D. Wren. 404 not found: the stability and persistence of URLs published in MEDLINE. *Bioinformatics*, 20(5):668–672, 2004.

[93] Chorng-Guang Wu, James H. Gerlach, and Clifford E. Young. An empirical analysis of open source software developers motivations and continuance intentions. *Information & Management*, 44(3): 253–262, April 2007. doi: 10.1016/j.im.2006.12.006.