How scientific software users choose what they use: preliminary results from an empirical survey

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

February 12, 2016

Abstract

When they seek software for a task, how do scientists go about finding it? Past research and anecdotal evidence suggest that searching the web, asking colleagues and reading papers have been the primary approaches used—but is that still true today, given the increasingly greater use of social media and socially-centric online systems such as StackOverflow and GitHub? In addition, when they *do* look for software, what are some of the main characteristics that influence scientific users' choice of software? And finally, if a systematic catalog of software were to be developed to help people find software, what kind of information would they like to see in it? These are the questions that motivated a survey we conducted in late 2015. The survey was designed to capture current practices and experiences in searching for software from two distinct groups: those looking for ready-to-run software and those looking for software source code. We present the results of our survey and discuss our findings in this report.

1. Introduction

Despite that software is critical to most research [5, 6, 20, 21, 51], finding software suitable for a given purpose remains surprisingly difficult [7, 10, 43]. Few effective resources exist to help users discover alternatives or understand the differences between them. The extent of the problem was made clear in a recent study of developers at Google, Inc.; the authors found that the factor "most disruptive to the [software] reuse process" was "difficulties in finding artifacts" [5]. In other words, *even the developers at Google have difficulty finding software.*

When asked, many people reply that they look for software by searching the web with a general-purpose search engine such as Google. Despite its popularity, this approach suffers from significant problems: Web searches can yield dozens or hundreds of viable candidates – and millions of irrelevant results. Moreover, some questions cannot be answered through search without substantial additional effort, such as what are the *specific* characteristics of different software tools or how do tools *differ* from each other. Many scientists also turn to the scientific literature to learn what others have used for similar tasks or research domains. Searching the literature can increase the relevance of results and provide other useful information, but it suffers from limitations too: publications are static documents which can take months or years to produce and may not reflect a tool's current capabilities [52], and moreover, not all software tools are mentioned in publications or have a publication associated with them. (As an example, the recent article announcing the first-ever detection of gravitational waves lacked any mention of any software used [1].) This may be for various reasons, such as article length limits or a disinclination on the part of researchers to describe their full software stack or workflow. Still other potential methods for finding software include asking colleagues, asking on social media, following institutional guidelines for recommended software, 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 [10]. We are interested in developing better resources to help users, particularly scientific users, discover software. In order to gain a better understanding of the factors that influence how software users locate software, we developed and distributed an electronic survey beginning in September, 2015, which we advertised to numerous mailing lists serving communities in astronomy and systems biology. We report on our methods, the survey responses, and our analyses below.

2. Survey structure

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 similar surveys in computing [e.g., 49], we iterated on the design of the instrument, paying 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 respondents' place of work.

To help iterate on the design of the survey instrument, we performed a pilot survey with close 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 logic so that the final number of questions actually seen by any given respondent depended on the answers selected to certain key 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 respondents 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. Sampling plan

We used nonprobabilistic convenience sampling with self-selection. We advertised the survey on mailing lists and social media oriented to the astronomical and biological sciences. The number of potential participants is unknown, because we cannot track the redistribution of the survey invitation. Consequently, the response rate is unknown.

Potential biasing factors in the results are the same as those common to most self-selected written surveys that use convenience sampling. These include response bias (i.e., people who responded may have different characteristics than those who did not), coverage errors (i.e., the representation of respondents 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 we (the authors) are relatively well-known within the subcommunities to which we advertised the survey, and thus the respondents may have been influenced by their personal knowledge or relationships to us.

2.3. Administration

We used Google Forms [19] to implement the survey instrument. The version of Google Forms was the free edition made available by Google, Inc. in the second half of 2015. Prior to making the survey form public and inviting participation in the survey, we obtained approval for the survey protocol from the California Institute of Technology's Committee for the Protection of Human Subjects (IRB). The survey form itself included a Web link to an electronic version 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 questions were optional.

On September 1, 2012, we used electronic mail to invite participation in the survey. As mentioned above, we advertised the survey on mailing lists and social media oriented to the astronomical and biological sciences, particularly to computational subcommunities within those domains. Any recipients were free to participate if they chose. The introduction and instructions for the survey were brief. The survey itself gave the motivations as "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." The instructions were given in email messages sent to mailing lists, and consisted of variations on the following: "As part of an NSF-funded project to prototype a software cataloguing system, we are running a survey to assess how people find software today. We very much welcome the input of fellow [*community*] people in this survey. Here is a link to the electronic form: [*URL*]".

The survey had no express closing date. We analyzed the results obtained by December 31, 2015. In total, we received 69 individual responses by that date. Responses were stored in a spreadsheet generated automatically by the survey system.

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 respondents

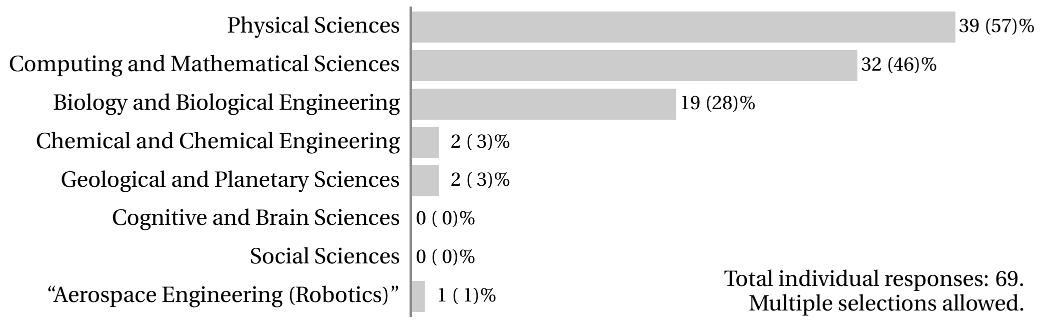


Figure 1: Respondents by discipline. The survey offered the first eight predefined categories and an additional slot for free text under "Other". Choices were nonexclusive. In some cases, respondents provided values for "Other" but the values were subsumed by one of the predefined categories; in those 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.

made use of this feature: 17 respondents selected two fields of work, six selected three fields, and one indicated four fields of work.

To enable us to gauge 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. 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 respondents regularly spent more than four hours of their day engaged with software, and 68% more than six hours.

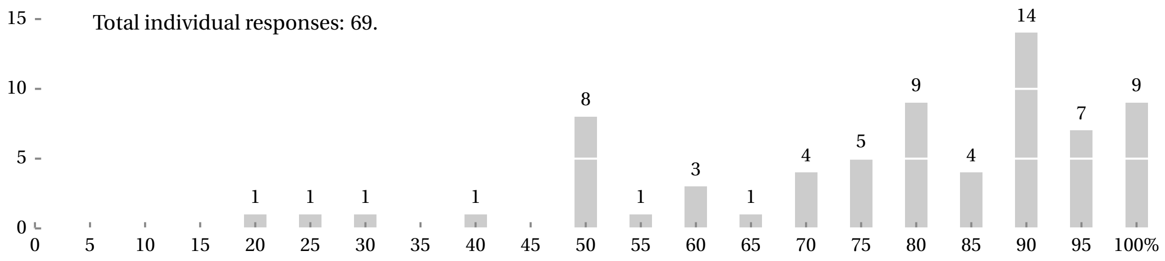


Figure 2: Bar graph of responses to the question "In your work, on a typical day, approximately what fraction of your time involves using or interacting directly with software on a computer or other computing device?"

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 they 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, and consequently all 69 respondents were shown the next set of questions in the survey.

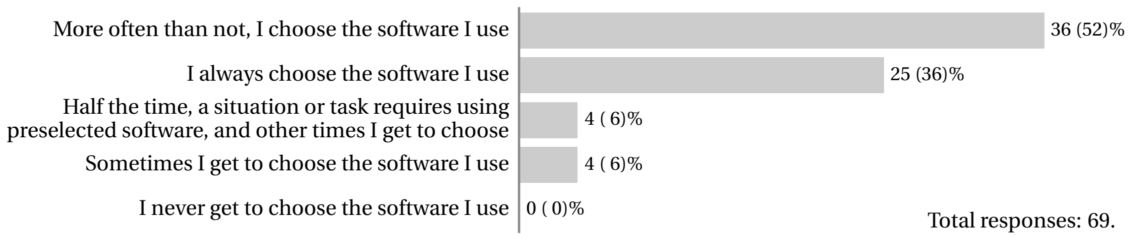
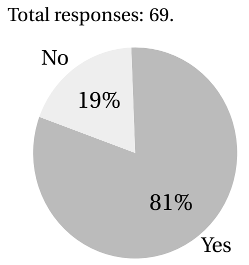


Figure 3: Responses to "In your work, how much freedom do you usually have to choose the software you use?".

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 these additional demographic questions or the questions relevant to developers, and were instead taken to the final survey feedback page. This happened to 13 respondents.) The first developer-related demographic question 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 on the next page provides a histogram of the responses received for those who answered the question with a usable answer (55 out of 56).



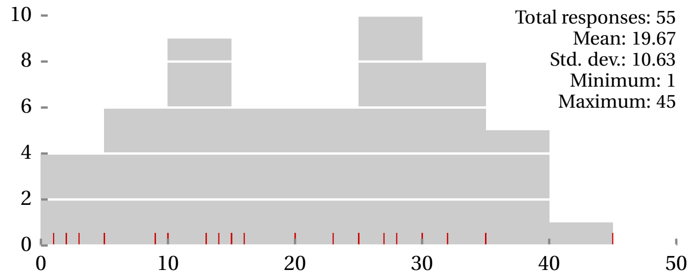


Figure 4: (Left) Responses to the question "Are you involved in software development?" (Right) Histogram and rug plot of years that respondents have been developing software (for those who also answered "Yes" to the question of whether they were involved in software development).

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?" The answers were in the form of 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, respondents were free to choose more than one and the explanatory text for the question indicated "If it is hard to identify a single one, you can indicate more than one below." Figure 5 provides a tally of the responses.

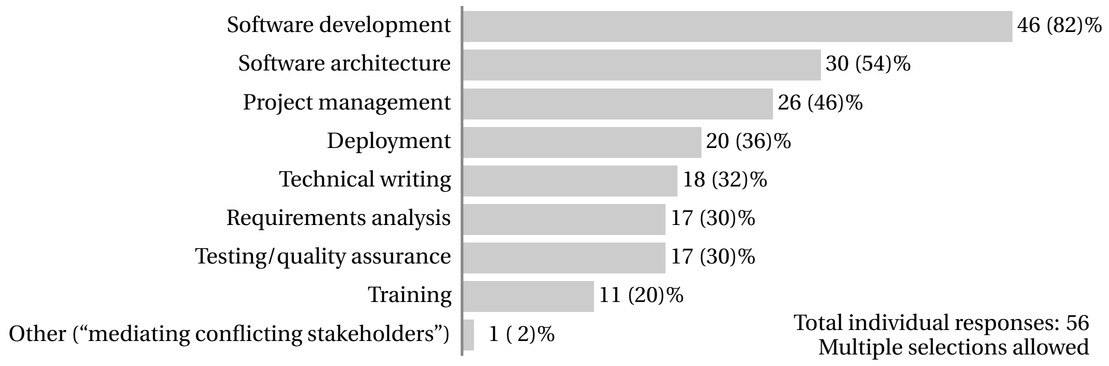


Figure 5: Responses to the question "In your current (or most recent) software development project, what is (or was) your primary responsibility?" This question was shown only to the 56 respondents who answered "Yes" to the question of whether they were involved in software development. This survey question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

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 click 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, and none of the respondents selected "Other". Figure 6 provides a summary of the results.

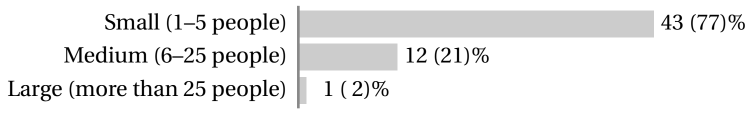


Figure 6: Project sizes reported by the 56 respondents who indicated they were involved in software development.

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 language choices as multiple 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 respondents), 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. Most respondents indicated they are involved in software development, and typical development team sizes were small, with 77% being in groups of 1 to 5 persons. This is common in scientific software development, 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.

Amongst the 81% of respondents who subsequently indicated that they were involved in software development to some degree (and not just end users), the median number of years of software development experience was 20. This also suggests that the typical respondent is mid-career or part of the pre-mobile device computing generation. In addition to software development per se, 65% indicated that they were also primarily responsible for project management or software architecture, which are traditionally more senior roles. The demographic data may thus indicate a possible bias in responses against more junior members of the respective communities, such as students and postdocs. This is of concern because junior members may have different search criteria and development experiences than their more experienced colleagues. This possibility should be borne in mind when interpreting the survey results. The cause of this distribution is unknown. We speculate that it may be the result of 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 we should aim to redress in future similar efforts.

4. Results: how respondents find software

Numbers in parentheses in this section indicate the relative rankings or levels of importance (essential and above-average) given in the survey.

4.1. Ready-to-use software

We first consider the search for software to use for a particular task rather for development purposes. This does not necessarily 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 a consideration even in this circumstance, and in the second question below, we sought 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 specific, predefined answer options were developed based on our own experiences as well as the similar questions posed in other surveys [2, 27, 39] and the results of our pilot run. Respondents were free to choose more than one answer. Figure 7 on the following page summarizes the results.

Personal recommendations (83%) and general search engines (90%) are the two main mechanisms employed to find software by our respondents. If a similar task is also described in the literature, then this will be used as a guide (62%) to identify relevant software to use. Online forums, whether general social media sites or more focused in scope, are an underutilized resources (mean of 21%), although people are slightly more likely to search in a public software project repository (32%) but not in a domain specific one (10%). This latter usage pattern may reflect ignorance of the existence of the topical indexes in question, but may also reflect a belief that such resources are too narrowly focused in scope for their needs. Unfortunately, the question did not address the size of the task being searched for so we cannot answer this for sure.

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 [14] 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 versions of this survey should include this as a predefined answer choice.

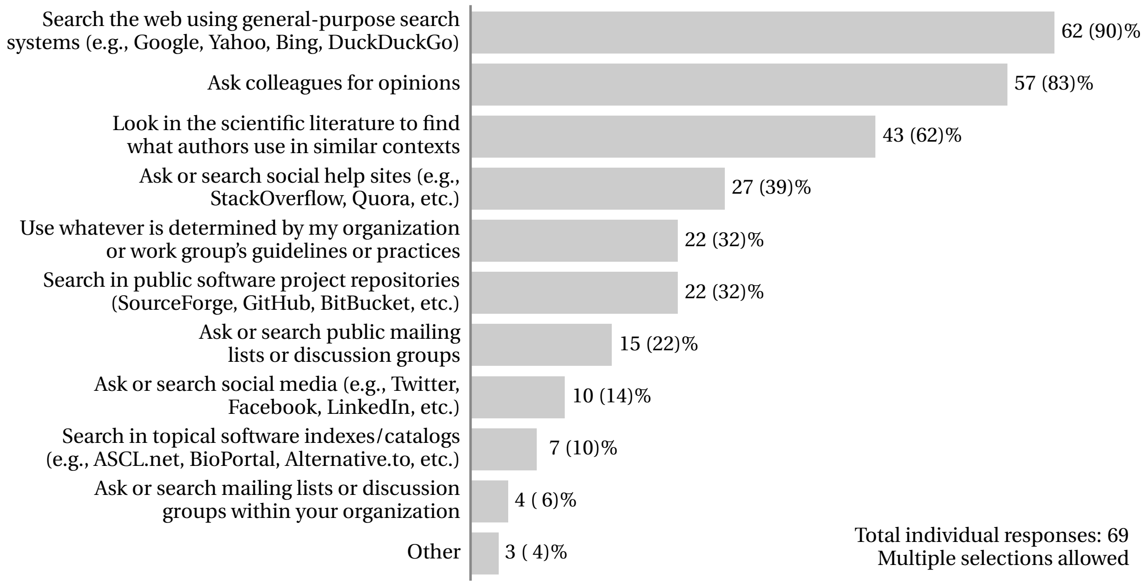


Figure 7: Responses to the question "When you need to find ready-to-run software for a particular task, how do you go about finding software?" Answer choices were nonexclusive. All 69 survey participants answered the question.

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. To this end, we posed the question "In general, how important are the following characteristics when you are searching for ready-to-run software for a task?" For the answer options, we provided a two-dimensional grid with different predefined criteria as the rows, and values on a unipolar rating scale for the columns. The available values on the scale were "Rarely or never important", "Somewhat or occasionally important", "Average importance", "Usually of above-average importance", and "Essential". Figure 8 on the next page summarizes the results. We sorted the rows of the bar graph using sum of ratings for "Essential" and "Usually of above-average importance" for each criterion, to reveal the most robust trends in the ratings. It is worth noting is that not all respondents provided a value for every criterion in the table, 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).

Unsurprisingly, people reported that the primary search criterion is the availability of specific features (95%) in the software. Support for specific data standards and file formats (82%) and software price (81%) 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 formats (e.g, FITS in astronomy). Platform requirements, in terms of both operating system (69%) and hardware (47%), score more highly than either ease of usage (50%), installation (40%) or any performance metrics (39%). How the software was actually implemented in terms of programming language (23%) or a particular software architecture (14%) are unimportant, though. We hypothesize that this is related to the dominance of particular computing configurations within specific sciences; for example, astronomical computing happens predominantly on Apple or Linux systems. In that context, if one is searching for software to run, a more pressing constraint may be simply be whether it is compatible with the operating platform rather than how it was written.

Quality and support aspects of the software are secondary considerations as far as software search is concerned, with documentation (48%) rating higher than either the reputation of the developer (27%) or the level of software support (18%). This suggests that once the software is installed, users expect that they can find in-house workarounds for any bugs, and that they are less concerned about future software updates. This illustrates how scientific computing is different from both commercial computing, where software licenses are purchased with implied support, and mobile computing, where software updates are automatically pushed out to devices.

A more surprising result is that other people's opinions of the software (23%) and its similarity to other software (12%) are not important search criteria when they are so important as search mechanisms (see above). It is possible that a search for software is only necessary when there are no word-of-mouth

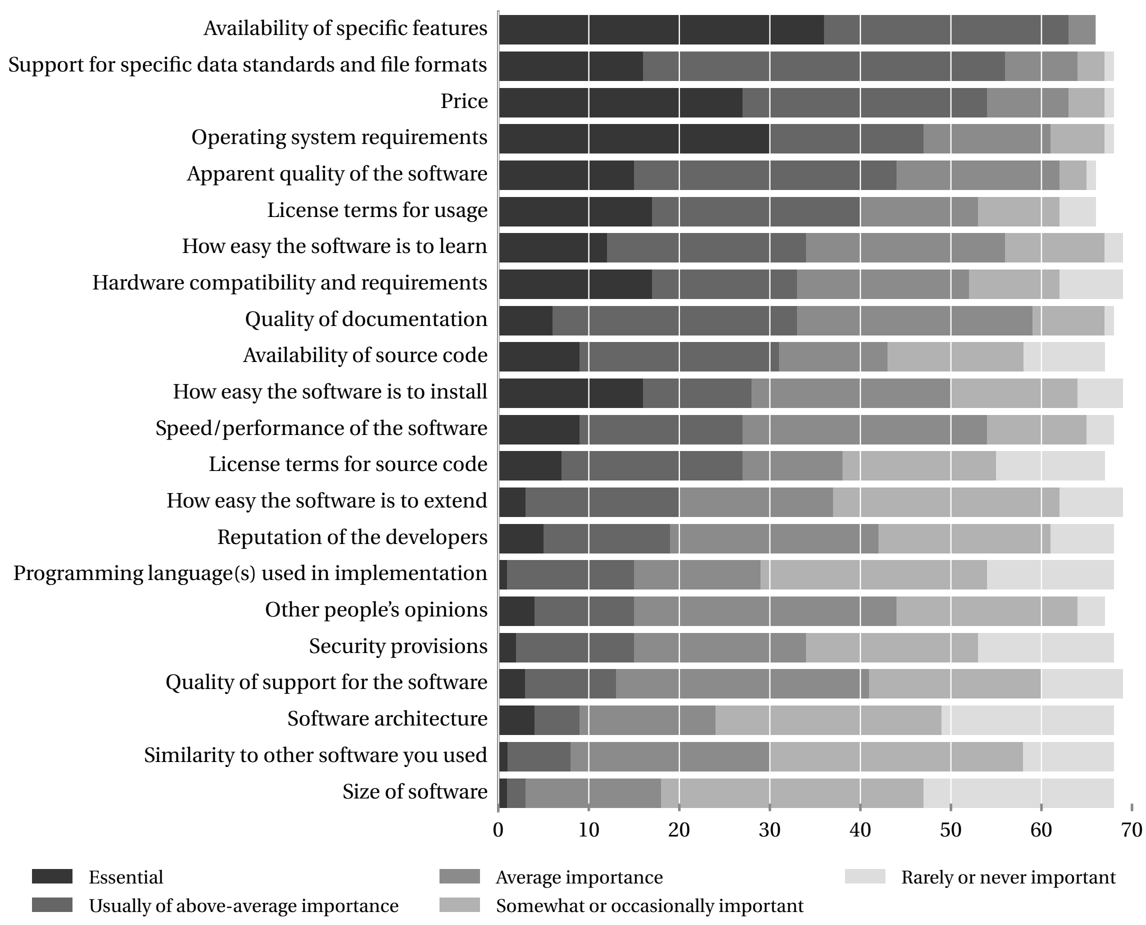


Figure 8: Responses to the question "In general, how important are the following characteristics when you are searching for ready-to-run software for a task?" All 69 respondents answered the question, but not all respondents chose to select an option for every possible characteristic. The bar graph is sorted by the sum of the number of times the options "Essential" and "Usually of above-average importance" were selected for each characteristic.

recommendations or literature recipes for particular tasks. 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.

Lastly, we should note that even though the question explicitly concerned ready-to-use software, the availability of the source code (46%) and its licensing (41%) are still relatively important considerations. This is probably a survey bias given the large fraction of respondees who identified as developers (to some degree). Such users are more likely to be willing to—and capable of—altering the software to meet their specific requirements, and the search criteria address this need.

4.1.3. Search case histories

To explore more deeply the topic of how users find software, we sought examples of respondents' past experiences by asking the open-ended question "(Optional) Please describe a past scenario when you looked for ready-to-run software." The survey form provided a text editing field where respondents 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 9 on the following page provides three examples taken from among those 14 responses.

Analysis of all responses to this question showed that the tasks covered a wide range of applications, ranging from authorization management software to MCMC samplers 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

— Sample A —

"Looked for a free UML modeler
Googling for some sofware 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 9: 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.

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 the 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 respondent 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 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

We now the consider the search for software source code. As described in Section 3 on page 3, our survey included the question "Are you involved in software development?" Responses were used to control whether the survey system showed additional questions related to software development. One of those questions 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 additional questions related to searching for source code. In this section, we discuss the results of that part of the survey.

4.2.1. Motivations

Out of the overall 69 respondents, 55 (80%) of them indicated they searched for source code at least some of the time. Figure 10 summarizes the responses received to the question "How often do you search online for software source code?" To help understand the motivations for why people search for software, we also asked the 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 11 summarizes the responses.

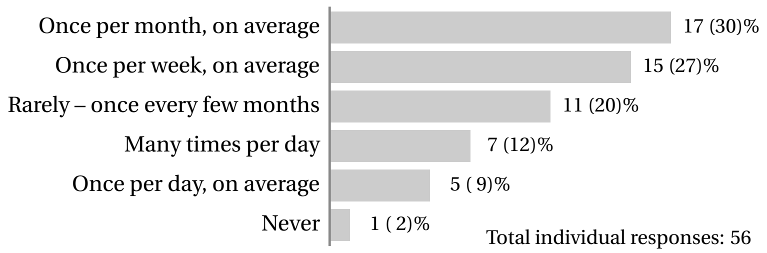


Figure 10: Responses to the question "How often do you search online for software source code?" Answer choices were presented as the mutually-exclusive multiple choices shown on the vertical axis.

The results show that our sample of developers typically search online for source code at least once a month (78%) with some (21%) searching at least once a day. Coding by example (76%) or reusing existing code as-is (72%) are the strongest motivations for searching. This indicates a clear preference for not reinventing the wheel or, at least, a reluctance to start from an absolutely blank slate when developing. Software refactoring is another common reason for searching, either to find a more efficient approach to an existing task (42%) or to discover new algorithms and data structures (36%). The web is also used as an online reference source to recall syntactic details (44%) or learn unfamiliar concepts (46%).

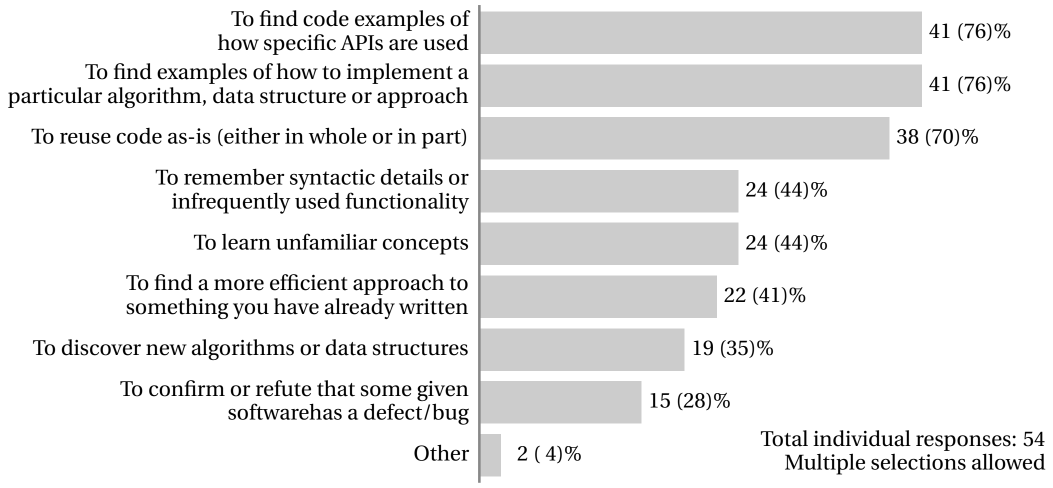


Figure 11: Responses to the question "What are some reasons why you look for source code (when you do look)?" This question offered the first eight predefined categories and an additional slot for free text under "Other". Answer choices were nonexclusive.

Two respondents wrote answers in the "Other" field: one wrote "Find software libraries to use", and the other, "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 survey 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. This question was similar to the earlier question "When you need to find ready-to-run software for a particular task, how do you go about finding software?" but included different answer options that are more relevant to searching for source code. Answer options were nonexclusive multiple choices, including an "Other" option with a field for free-text input. Figure 12 provides a summary of the results. The question was answered by all 55 respondents who indicated they searched for source code on at least some occasions.

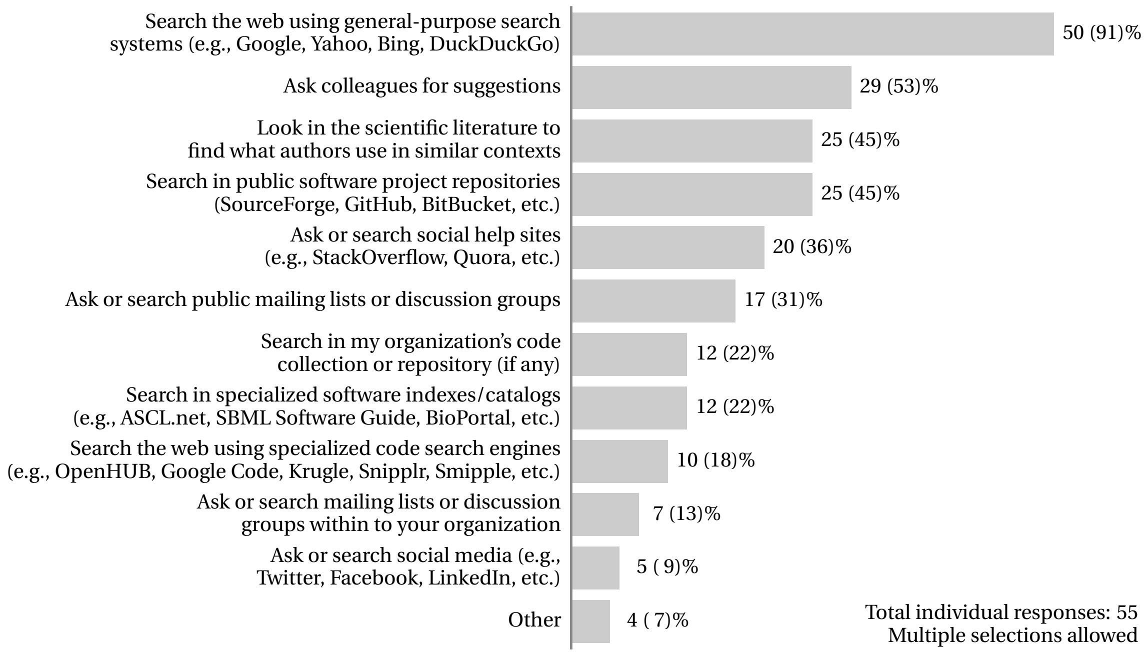


Figure 12: Responses to the question "What are some approaches you have used to look for source code in the past?" This question offered the first eleven predefined categories and an additional slot for free text under "Other". Answer choices were nonexclusive. A total of 55 respondents answered this question.

General-purpose search engines (91%) remain the primary mechanism used to search for source code but personal recommendations (54%) and the scientific literature (45%) are employed significantly less than when looking for ready-to-use software (Section 4.1.1 on page 6). This suggests that software development is a more subjective activity and so source code searches have a more personalized signature, determined in part by personal coding styles. (This may have implications for search anonymity issues.) This view is supported by the two main factors identified for the failure of past code searches (see below): an inability to find any code suitable for purpose (69%) and a belief that the requirements were too unique (65%).

The results also indicate that online forums remain underutilized (with a mean of 23%) but there is slightly greater use of public code repositories (46%), although not significantly so. Specialized code search engines (19%) and software indexes (22%) are also less used than expected but this may again reflect a lack of knowledge of their existence or coverage. It may also be another instance of sample bias with a more mature audience being less likely to use or trust newer domain-specific software, preferring instead more proven general-purpose mechanisms. 24% cited lack of trust in the options found as the reason for a previous failed search result.

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 thes 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.

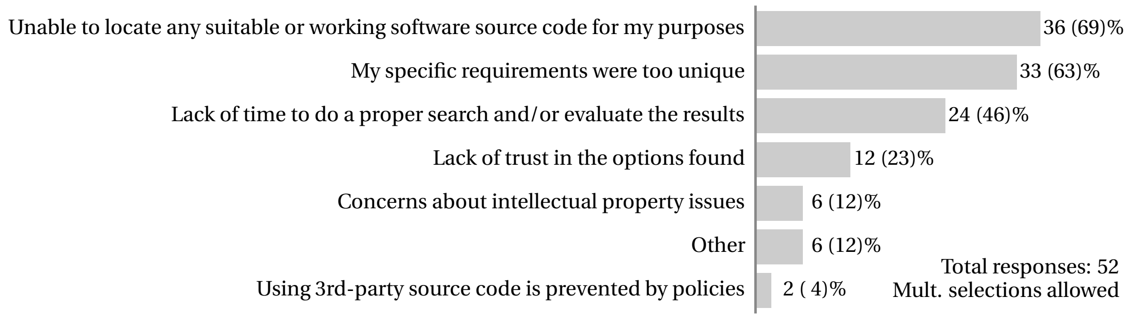


Figure 13: Responses to the question "What are some factors that have hindered your ability to FIND source code in the past?" This question offered the first six predefined categories and an additional slot for free text under "Other". Answer choices were nonexclusive. A total of 51 survey respondents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

There is a reasonable indication (46%) that time limitations often impact the ability to conduct proper searches for source code or to evaluate the results. This is may be due to the large number of results that general-purpose search engines can return, which in turn may make it difficult to find suitable results easily. This can be particularly the case if the ranking system of the search engine is optimizing 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 on page 6), 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.

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 respondents 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 next 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%) or install (29%) 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

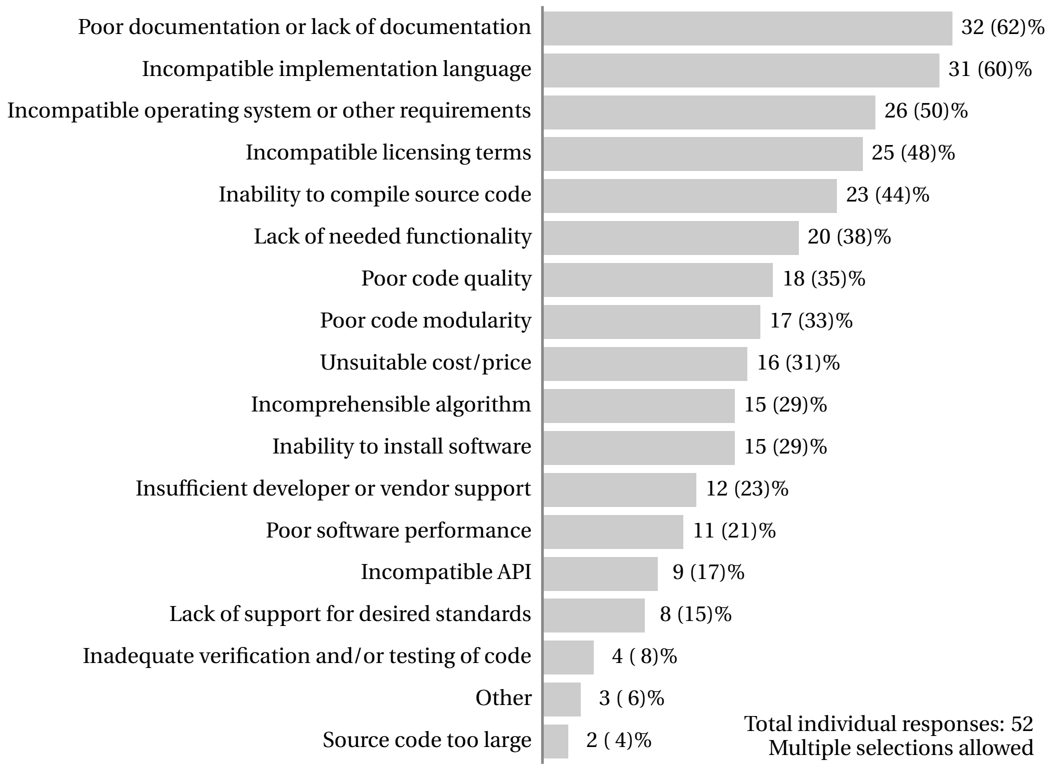


Figure 14: Responses to the question "If you searched and found source code in the past, what are some factors that may have prevented you from REUSING the source code you found?" This question offered the first 17 predefined categories and an additional slot for free text under "Other". Answer choices were nonexclusive. A total of 52 survey respondents answered this question.

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 you just want to learn something.

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: desirable metadata about software

A potential aid to finding software is a software catalog or index that would classify known software and allow people to browse and search for software by various criteria [28]. As part of our survey, we sought to determine what kind of information would be important to record about each entry in such an index. We therefore posed a question of all respondents who indicated they had the freedom to choose software. (In other words, not only those who indicated they developed software.) The question was, "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 respondents to our survey replied to this question. Figure 15 on the following page summarizes the results.

The results indicate that simple terms related to functional aspects – the name (88%), purpose (93%), domain of application (78%), data formats supported (76%) – and operational aspects – operating system(s)

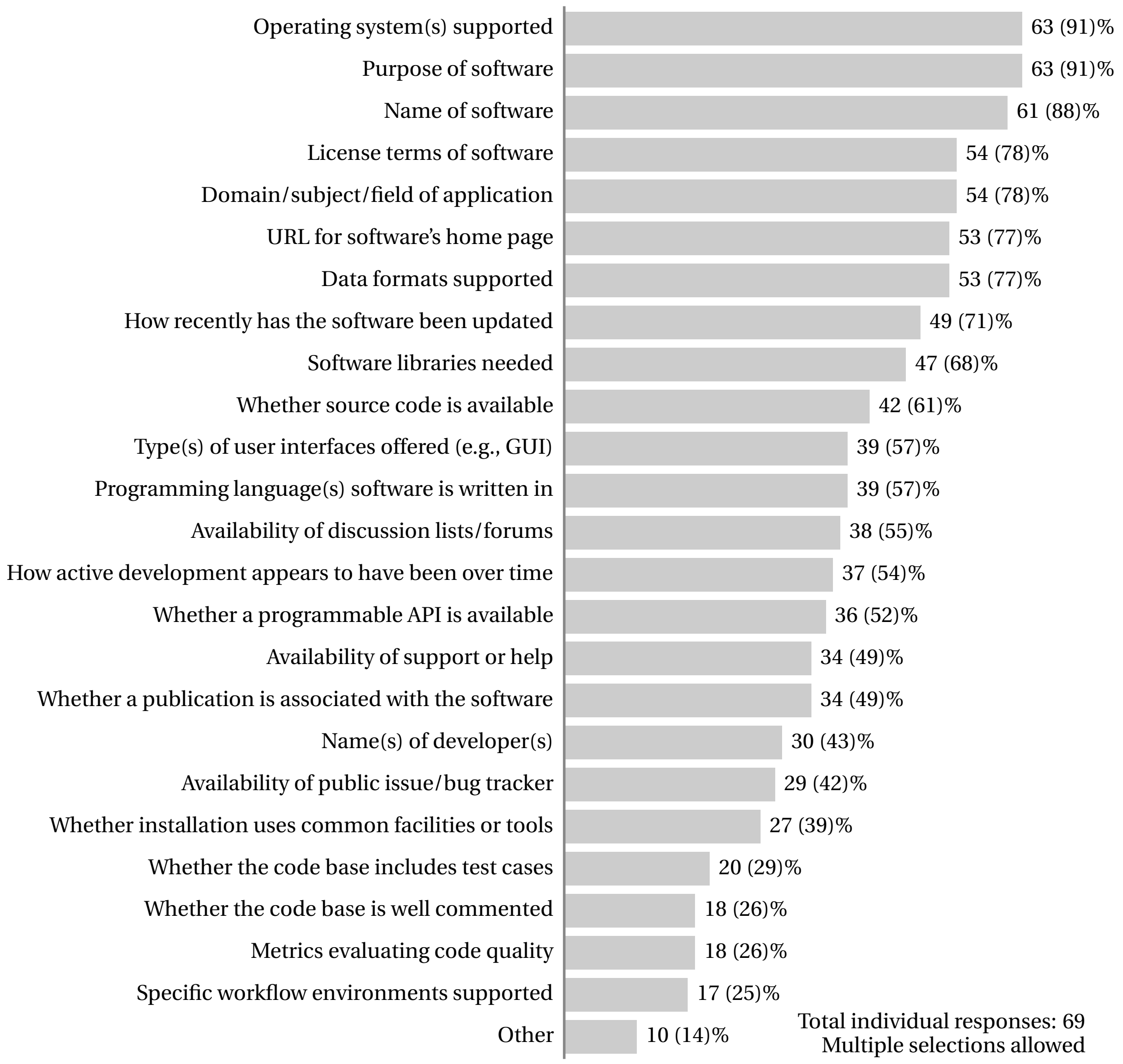


Figure 15: Responses to the question "Suppose that it were possible to create a public, searchable catalog or index of software, one that would record information about software of all kinds found anywhere. What kind of information would you find most useful to include for each entry in such a catalog or index?" This question offered the first 24 predefined categories and an additional slot for free text under "Other". Answer choices were nonexclusive. A total of 69 survey respondents answered this question.

(91%), data licensing (78%), and dependencies (69%) – scored most highly. Terms related to an increased degree of sophistication in these areas, such as the types of user interfaces offered (56%), a programmable API (51%), or if the installation process made use of common tools (40%), were ranked less highly. The availability of documentation, either online (78%), in online forums (54%), or in an associated paper (50%), was also an relatively important criterion, although the availability of support (49%) or bug tracking (41%) continue not to be so.

Implementation details, such as the availability of source code (62%), which language(s) was used (57%), and the name of the developer (43%), would be more important to developers. Again, however, more formal aspects – test cases (28%), well-commented code (26%) or metrics evaluating code quality (26%) – rank low. It should be noted that the development status of any code (53%) and when it was last updated (71%) rate more highly. This suggests that there is a preference for current software rather than legacy or dead projects, even if they match other search requirements.

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, but the remaining write-in values constitute sufficiently different categories of information that they are 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 lengthy literature survey seeking 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 numerous past 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 two reports of surveys that were not focused specifically on a software development context [22, 25].

Joppa et al. [25] 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 underlying our survey question "When you need to find ready-to-run software for a particular task, how do you go about finding software?" In order of most popular to least, the answers that mention 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 [25] 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 web searches.

Huang et al. [22] summarize interviews of 15 researchers working in bioinformatics. Their subjects included students and full-time faculty. The authors reported four factors influenced the selection of scientific software: (1) suggestions given by mentors or senior members of an institution; (2) mentor involvement in the development of the software, in cases where mentors are also developers; (3) the number of publications *about* some software; and (4) the reputation of the software, assessed by counting the number of publications mentioning the *use* of the software. Huang et al.’s report [22] does not include any quantitative or qualitative data about the relative importance of these factors.

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 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 the next section.

One of the earliest surveys of software developers in the context of software reuse was published by Frakes and Fox [13]. 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 of their questions and results are relevant to the topics of our own survey:

- On the topic of whether reuse is more common in some industries than others, they found that indeed it is. The subjects in Frakes and Fox’s study [13] 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.

- On the topic of whether having a repository improved reuse, they found that it did not. 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 respondents prefer to develop their own software versus reusing someone else’s, Frakes and Fox found that 72% of respondents 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 respondents (but this was because the level of quality they encountered had been adequate).

Samadi et al. [34] report 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".
- On the topic of the criteria used by people 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 same study was reprised in 2005 with a wider audience that included members of academia. The survey and the results of 100 responses they received are summarized by Marshall et al. [28]. There were four groups of questions: background information, recent reuse experiences, reuse development practices, and community needs. (The actual questions and detailed responses are not provided in the paper.) According to Marshall et al., the larger 2005 survey produced essentially similar results to the preliminary 2004 survey. The 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 respondents 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. [28] 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 [30] studied 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 reused. Unfortunately, the Orrego and Mundy did not report the specific approaches attempted by people to locate software.

Singer et al. [42] examined how software developers active on GitHub use Twitter. They conducted an initial exploratory survey with 271 users of GitHub users (270 of who 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 learn 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. [5] 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 [5] 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%).
- They asked about the preferred ways to find reusables. 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 system features a centralized code search facility.
- They also asked, "What do you do to properly understand and adequately select reusable artifacts" yielded 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%).

7.3. Surveys examining code search by developers

It has been long known from studies of developers at work [e.g., 41] 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. [45, 46] surveyed Java programmers in 2006–2007 to understand how and why they searched for source code. They solicited participation to fill out a web survey via invitations to mailing lists and newsgroups, 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. [40] 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 searchers described, Umarji et al. [46] 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 respondent, "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 [45] 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 [47] (49%), (4) references from peers (43%), (4) mailing lists (23%), and (5) code-specific search engines such as Google Code Search [18] (16%).

- With respect to the selection criteria used by developers to choose a solution, Umarji and Sim [45] report that the most important factors in order of importance were: (1) 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 [15] 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. [36] performed a pair of surveys as part of an effort to understanding 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 in the different surveys. 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 respondents 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. [33] 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.

- They 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 [2, 3, 8, 9, 17, 24, 26, 44, 50], and observational studies (often coupled with interviews), either in institutional settings or in laboratory environments [4, 9, 11, 16, 29, 32, 35, 37–39]. 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 [12] 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 scientific users as well as the criteria that people apply to select between alternative software choices. Our respondents 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 self-selection of respondents and convenience sampling used in our survey means we cannot draw quantitative conclusions about the larger population; however, our survey's results are similar to those of other past surveys, both in scientific communities and in the broader commercial world where the question topics overlap, and this gives us reason to believe that our results are qualitatively representative of the subcommunities we sampled.

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 in order of popularity are: (i) search the web with general-purpose search engines, (ii) ask colleagues, (iii) look in the scientific literature, (iv) look in social help sites such as StackOverflow, and (v) use whatever is determined by the guidelines or work practices of the group or organization where one is working.

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 StackOverflow.

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 desirable by at least 50% of the respondents, with operating system support, purpose of software, name of software, license terms, and domain/field of application being the most-often requested features.

These have implications for development of catalogs or other resources to help people find software.

9. Acknowledgments

This work was funded by the USA National Science Foundation, award #1533792. The figures in this paper were generated using `matplotlib` [23] version 1.5.1, using programs written in the language Python [31, 48], version 3.4.

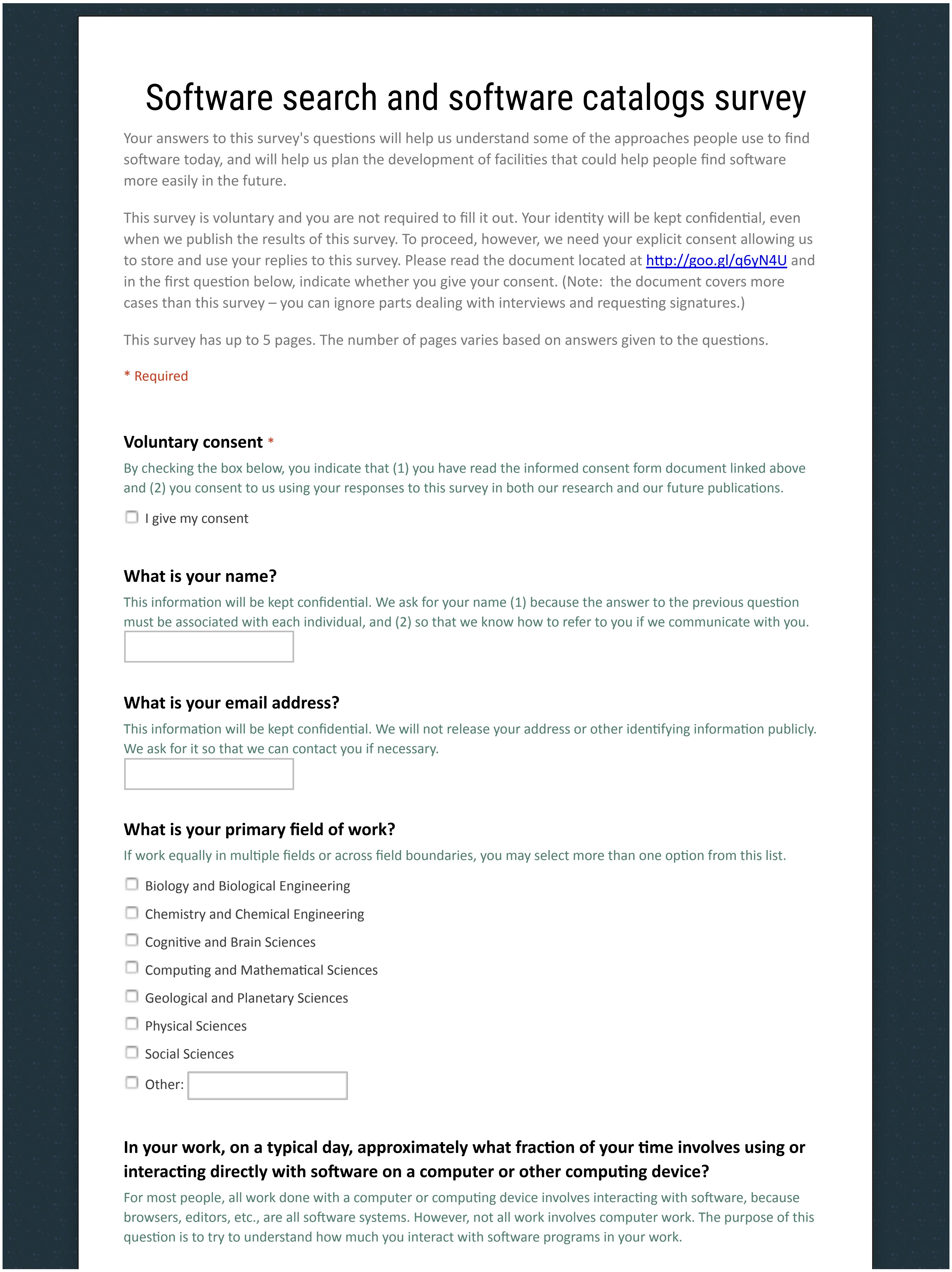
Appendix A. The Survey

We implemented the survey using Google Forms [19], 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

Powered by Google Forms

This content is neither created nor endorsed by Google.

Report Abuse - Terms of Service - Additional Terms

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

Powered by Google Forms

This content is neither created nor endorsed by Google.

Report Abuse - Terms of Service - Additional Terms

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

Powered by
Google Forms

This content is neither created nor endorsed by Google.
Report Abuse - Terms of Service - Additional Terms

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:

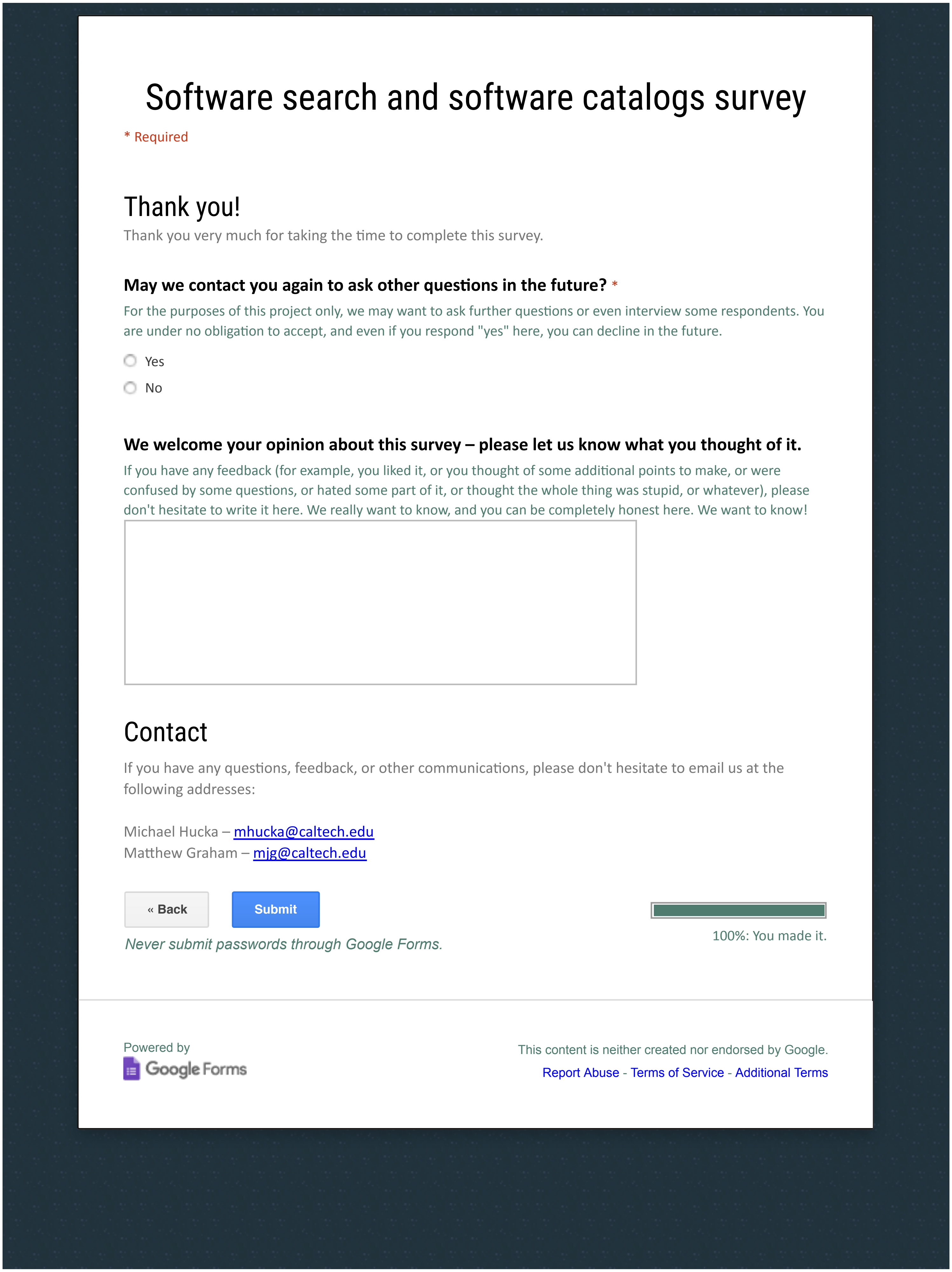
« Back   Continue »

80% completed

Powered by Google Forms

This content is neither created nor endorsed by Google.

Report Abuse - Terms of Service - Additional Terms



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

Never submit passwords through Google Forms.

100%: You made it.

Powered by Google Forms

This content is neither created nor endorsed by Google.
Report Abuse - Terms of Service - Additional Terms

References

[1] B. P. Abbott, R. Abbott, T. D. Abbott, M. R. Abernathy, F. Acernese, K. Ackley, C. Adams, T. Adams, P. Addesso, R. X. Adhikari, V. B. Adya, C. Affeldt, M. Agathos, K. Agatsuma, N. Aggarwal, O. D. Aguiar, L. Aiello, A. Ain, P. Ajith, B. Allen, A. Allocca, P. A. Altin, S. B. Anderson, W. G. Anderson, K. Arai, M. A. Arain, M. C. Araya, C. C. Arceneaux, J. S. Areeda, N. Arnaud, K. G. Arun, S. Ascenzi, G. Ashton, M. Ast, S. M. Aston, P. Astone, P. Aufmuth, C. Aulbert, S. Babak, P. Bacon, M. K. M. Bader, P. T. Baker, F. Baldaccini, G. Ballardin, S. W. Ballmer, J. C. Barayoga, S. E. Barclay, B. C. Barish, D. Barker, F. Barone, B. Barr, L. Barsotti, M. Barsuglia, D. Barta, J. Bartlett, M. A. Barton, I. Bartos, R. Bassiri, A. Basti, J. C. Batch, C. Baune, V. Bavigadda, M. Bazzan, B. Behnke, M. Bejger, C. Belczynski, A. S. Bell, C. J. Bell, B. K. Berger, J. Bergman, G. Bergmann, C. P. L. Berry, D. Bersanetti, A. Bertolini, J. Betzwieser, S. Bhagwat, R. Bhandare, I. A. Bilenko, G. Billingsley, J. Birch, R. Birney, O. Birnholtz, S. Biscans, A. Bisht, M. Bitossi, C. Biwer, M. A. Bizouard, J. K. Blackburn, C. D. Blair, D. G. Blair, R. M. Blair, S. Bloemen, O. Bock, T. P. Bodiya, M. Boer, G. Bogaert, C. Bogan, A. Bohe, P. Bojtos, C. Bond, F. Bondu, R. Bonnand, B. A. Boom, R. Bork, V. Boschi, S. Bose, Y. Bouffanais, A. Bozzi, C. Bradaschia, P. R. Brady, V. B. Braginsky, M. Branchesi, J. E. Brau, T. Briant, A. Brillet, M. Brinkmann, V. Brisson, P. Brockill, A. F. Brooks, D. A. Brown, D. D. Brown, N. M. Brown, C. C. Buchanan, A. Buikema, T. Bulik, H. J. Bulten, A. Buonanno, D. Buskulic, C. Buy, R. L. Byer, M. Cabero, L. Cadonati, G. Cagnoli, C. Cahillane, J. Calderón Bustillo, T. Callister, E. Calloni, J. B. Camp, K. C. Cannon, J. Cao, C. D. Capano, E. Capocasa, F. Carbognani, S. Caride, J. Casanueva Diaz, C. Casentini, S. Caudill, M. Cavaglià, F. Cavalier, R. Cavalieri, G. Cella, C. B. Cepeda, L. Cerboni Baiardi, G. Cerretani, E. Cesarini, R. Chakraborty, T. Chalermsongsak, S. J. Chamberlin, M. Chan, S. Chao, P. Charlton, E. Chassande-Mottin, H. Y. Chen, Y. Chen, C. Cheng, A. Chincarini, A. Chiummo, H. S. Cho, M. Cho, J. H. Chow, N. Christensen, Q. Chu, S. Chua, S. Chung, G. Ciani, F. Clara, J. A. Clark, F. Cleva, E. Coccia, P.-F. . Cohadon, A. Colla, C. G. Collette, L. Cominsky, M. Constancio, A. Conte, L. Conti, D. Cook, T. R. Corbitt, N. Cornish, A. Corsi, S. Cortese, C. A. Costa, M. W. Coughlin, S. B. Coughlin, J.-P. . Coulon, S. T. Countryman, P. Couvares, E. E. Cowan, D. M. Coward, M. J. Cowart, D. C. Coyne, R. Coyne, K. Craig, J. D. E. Creighton, T. D. Creighton, J. Cripe, S. G. Crowder, A. M. Cruise, A. Cumming, L. Cunningham, E. Cuoco, T. Dal Canton, S. L. Danilishin, S. DAntonio, K. Danzmann, N. S. Darman, C. F. Da Silva Costa, V. Dattilo, I. Dave, H. P. Daveloza, M. Davier, G. S. Davies, E. J. Daw, R. Day, S. De, D. DeBra, G. Debreczeni, J. Degallaix, M. De Laurentis, S. Deléglise, W. Del Pozzo, T. Denker, T. Dent, H. Dereli, V. Dergachev, R. T. DeRosa, R. De Rosa, R. DeSalvo, S. Dhurandhar, M. C. Díaz, L. Di Fiore, M. Di Giovanni, A. Di Lieto, S. Di Pace, I. Di Palma, A. Di Virgilio, G. Dojcinoski, V. Dolique, F. Donovan, K. L. Dooley, S. Doravari, R. Douglas, T. P. Downes, M. Drago, R. W. P. Drever, J. C. Driggers, Z. Du, M. Ducrot, S. E. Dwyer, T. B. Edo, M. C. Edwards, A. Effler, H.-B. . Eggenstein, P. Ehrens, J. Eichholz, S. S. Eikenberry, W. Engels, R. C. Essick, T. Etzel, M. Evans, T. M. Evans, R. Everett, M. Factourovich, V. Fafone, H. Fair, S. Fairhurst, X. Fan, Q. Fang, S. Farinon, B. Farr, W. M. Farr, M. Favata, M. Fays, H. Fehrmann, M. M. Fejer, D. Feldbaum, I. Ferrante, E. C. Ferreira, F. Ferrini, F. Fidecaro, L. S. Finn, I. Fiori, D. Fiorucci, R. P. Fisher, R. Flaminio, M. Fletcher, H. Fong, J.-D. . Fournier, S. Franco, S. Frasca, F. Frasconi, M. Frede, Z. Frei, A. Freise, R. Frey, V. Frey, T. T. Fricke, P. Fritschel, V. V. Frolov, P. Fulda, M. Fyffe, H. A. G. Gabbard, J. R. Gair, L. Gammaitoni, S. G. Gaonkar, F. Garufi, A. Gatto, G. Gaur, N. Gehrels, G. Gemme, B. Gendre, E. Genin, A. Gennai, J. George, L. Gergely, V. Germain, Abhirup Ghosh, Archisman Ghosh, S. Ghosh, J. A. Giaime, K. D. Giardina, A. Giazotto, K. Gill, A. Glaefke, J. R. Gleason, E. Goetz, R. Goetz, L. Gondan, G. González, J. M. Gonzalez Castro, A. Gopakumar, N. A. Gordon, M. L. Gorodetsky, S. E. Gossan, M. Gosselin, R. Gouaty, C. Graef, P. B. Graff, M. Granata, A. Grant, S. Gras, C. Gray, G. Greco, A. C. Green, R. J. S. Greenhalgh, P. Groot, H. Grote, S. Grunewald, G. M. Guidi, X. Guo, A. Gupta, M. K. Gupta, K. E. Gushwa, E. K. Gustafson, R. Gustafson, J. J. Hacker, B. R. Hall, E. D. Hall, G. Hammond, M. Haney, M. M. Hanke, J. Hanks, C. Hanna, M. D. Hannam, J. Hanson, T. Hardwick, J. Harms, G. M. Harry, I. W. Harry, M. J. Hart, M. T. Hartman, C.-J. . Haster, K. Haughian, J. Healy, J. Heefner, A. Heidmann, M. C. Heintze, G. Heinzel, H. Heitmann, P. Hello, G. Hemming, M. Hendry, I. S. Heng, J. Hennig, A. W. Heptonstall, M. Heurs, S. Hild, D. Hoak, K. A. Hodge, D. Hofman, S. E. Hollitt, K. Holt, D. E. Holz, P. Hopkins, D. J. Hosken, J. Hough, E. A. Houston, E. J. Howell, Y. M. Hu, S. Huang, E. A. Huerta, D. Huet, B. Hughey, S. Husa, S. H. Huttner, T. Huynh-Dinh, A. Idrisy, N. Indik, D. R. Ingram, R. Inta, H. N. Isa, J.-M. . Isac, M. Isi, G. Islas, T. Isogai, B. R. Iyer, K. Izumi, M. B. Jacobson, T. Jacqmin, H. Jang, K. Jani, P. Jaranowski, S. Jawahar, F. Jiménez-Forteza, W. W. Johnson, N. K. Johnson-McDaniel, D. I. Jones, R. Jones, R. J. G. Jonker, L. Ju, K. Haris, C. V. Kalaghatgi, V. Kalogera, S. Kandhasamy, G. Kang, J. B. Kanner, S. Karki, M. Kasprzack, E. Katsavounidis, W. Katzman, S. Kaufer, T. Kaur, K. Kawabe, F. Kawazoe, F. Kéfélian, M. S. Kehl, D. Keitel, D. B. Kelley, W. Kells, R. Kennedy, D. G.

Keppel, J. S. Key, A. Khalaidovski, F. Y. Khalili, I. Khan, S. Khan, Z. Khan, E. A. Khazanov, N. Kijbunchoo, C. Kim, J. Kim, K. Kim, Nam-Gyu Kim, Namjun Kim, Y.-M. . Kim, E. J. King, P. J. King, D. L. Kinzel, J. S. Kissel, L. Kleybolte, S. Klimenko, S. M. Koehlenbeck, K. Kokeyama, S. Koley, V. Kondrashov, A. Kontos, S. Koranda, M. Korobko, W. Z. Korth, I. Kowalska, D. B. Kozak, V. Kringel, B. Krishnan, A. Królak, C. Krueger, G. Kuehn, P. Kumar, R. Kumar, L. Kuo, A. Kutynia, P. Kwee, B. D. Lackey, M. Landry, J. Lange, B. Lantz, P. D. Lasky, A. Lazzarini, C. Lazzaro, P. Leaci, S. Leavey, E. O. Lebigot, C. H. Lee, H. K. Lee, H. M. Lee, K. Lee, A. Lenon, M. Leonardi, J. R. Leong, N. Leroy, N. Letendre, Y. Levin, B. M. Levine, T. G. F. Li, A. Libson, T. B. Littenberg, N. A. Lockerbie, J. Logue, A. L. Lombardi, L. T. London, J. E. Lord, M. Lorenzini, V. Loriette, M. Lormand, G. Losurdo, J. D. Lough, C. O. Lousto, G. Lovelace, H. Lück, A. P. Lundgren, J. Luo, R. Lynch, Y. Ma, T. MacDonald, B. Machenschalk, M. MacInnis, D. M. Macleod, F. Magaña Sandoval, R. M. Magee, M. Mageswaran, E. Majorana, I. Maksimovic, V. Malvezzi, N. Man, I. Mandel, V. Mandic, V. Mangano, G. L. Mansell, M. Manske, M. Mantovani, F. Marchesoni, F. Marion, S. Márka, Z. Márka, A. S. Markosyan, E. Maros, F. Martelli, L. Martellini, I. W. Martin, R. M. Martin, D. V. Martynov, J. N. Marx, K. Mason, A. Masserot, T. J. Massinger, M. Masso-Reid, F. Matichard, L. Matone, N. Mavalvala, N. Mazumder, G. Mazzolo, R. McCarthy, D. E. McClelland, S. McCormick, S. C. McGuire, G. McIntyre, J. McIver, D. J. McManus, S. T. McWilliams, D. Meacher, G. D. Meadors, J. Meidam, A. Melatos, G. Mendell, D. Mendoza-Gandara, R. A. Mercer, E. Merilh, M. Merzougui, S. Meshkov, C. Messenger, C. Messick, P. M. Meyers, F. Mezzani, H. Miao, C. Michel, H. Middleton, E. E. Mikhailov, L. Milano, J. Miller, M. Millhouse, Y. Minenkov, J. Ming, S. Mirshekari, C. Mishra, S. Mitra, V. P. Mitrofanov, G. Mitselmakher, R. Mittleman, A. Moggi, M. Mohan, S. R. P. Mohapatra, M. Montani, B. C. Moore, C. J. Moore, D. Moraru, G. Moreno, S. R. Morriss, K. Mossavi, B. Mours, C. M. Mow-Lowry, C. L. Mueller, G. Mueller, A. W. Muir, Arunava Mukherjee, D. Mukherjee, S. Mukherjee, N. Mukund, A. Mullavey, J. Munch, D. J. Murphy, P. G. Murray, A. Mytidis, I. Nardecchia, L. Naticchioni, R. K. Nayak, V. Necula, K. Nedkova, G. Nelemans, M. Neri, A. Neunzert, G. Newton, T. T. Nguyen, A. B. Nielsen, S. Nissanke, A. Nitz, F. Nocera, D. Nolting, M. E. N. Normandin, L. K. Nuttall, J. Oberling, E. Ochsner, J. ODell, E. Oelker, G. H. Ogin, J. J. Oh, S. H. Oh, F. Ohme, M. Oliver, P. Oppermann, Richard J. Oram, B. OReilly, R. OShaughnessy, C. D. Ott, D. J. Ottaway, R. S. Ottens, H. Overmier, B. J. Owen, A. Pai, S. A. Pai, J. R. Palamos, O. Palashov, C. Palomba, A. Pal-Singh, H. Pan, Y. Pan, C. Pankow, F. Pannarale, B. C. Pant, F. Paoletti, A. Paoli, M. A. Papa, H. R. Paris, W. Parker, D. Pascucci, A. Pasqualetti, R. Passaquieti, D. Passuello, B. Patricelli, Z. Patrick, B. L. Pearlstone, M. Pedraza, R. Pedurand, L. Pekowsky, A. Pele, S. Penn, A. Perreca, H. P. Pfeiffer, M. Phelps, O. Piccinni, M. Pichot, M. Pickenpack, F. Piergiovanni, V. Pierro, G. Pillant, L. Pinard, I. M. Pinto, M. Pitkin, J. H. Poeld, R. Poggiani, P. Popolizio, A. Post, J. Powell, J. Prasad, V. Predoi, S. S. Premachandra, T. Prestegard, L. R. Price, M. Prijatelj, M. Principe, S. Privitera, R. Prix, G. A. Prodi, L. Prokhorov, O. Puncken, M. Punturo, P. Puppo, M. Pürrer, H. Qi, J. Qin, V. Quetschke, E. A. Quintero, R. Quitzow-James, F. J. Raab, D. S. Rabeling, H. Radkins, P. Raffai, S. Raja, M. Rakhmanov, C. R. Ramet, P. Rapagnani, V. Raymond, M. Razzano, V. Re, J. Read, C. M. Reed, T. Regimbau, L. Rei, S. Reid, D. H. Reitze, H. Rew, S. D. Reyes, F. Ricci, K. Riles, N. A. Robertson, R. Robie, F. Robinet, A. Rocchi, L. Rolland, J. G. Rollins, V. J. Roma, J. D. Romano, R. Romano, G. Romanov, J. H. Romie, D. Rosińska, S. Rowan, A. Rüdiger, P. Ruggi, K. Ryan, S. Sachdev, T. Sadecki, L. Sadeghian, L. Salconi, M. Saleem, F. Salemi, A. Samajdar, L. Sammut, L. M. Sampson, E. J. Sanchez, V. Sandberg, B. Sandeen, G. H. Sanders, J. R. Sanders, B. Sassolas, B. S. Sathyaprakash, P. R. Saulson, O. Sauter, R. L. Savage, A. Sawadsky, P. Schale, R. Schilling, J. Schmidt, P. Schmidt, R. Schnabel, R. M. S. Schofield, A. Schönbeck, E. Schreiber, D. Schuette, B. F. Schutz, J. Scott, S. M. Scott, D. Sellers, A. S. Sengupta, D. Sentenac, V. Sequino, A. Sergeev, G. Serna, Y. Setyawati, A. Sevigny, D. A. Shaddock, T. Shaffer, S. Shah, M. S. Shahriar, M. Shaltev, Z. Shao, B. Shapiro, P. Shawhan, A. Sheperd, D. H. Shoemaker, D. M. Shoemaker, K. Siellez, X. Siemens, D. Sigg, A. D. Silva, D. Simakov, A. Singer, L. P. Singer, A. Singh, R. Singh, A. Singhal, A. M. Sintes, B. J. J. Slagmolen, J. R. Smith, M. R. Smith, N. D. Smith, R. J. E. Smith, E. J. Son, B. Sorazu, F. Sorrentino, T. Souradeep, A. K. Srivastava, A. Staley, M. Steinke, J. Steinlechner, S. Steinlechner, D. Steinmeyer, B. C. Stephens, S. P. Stevenson, R. Stone, K. A. Strain, N. Straniero, G. Stratta, N. A. Strauss, S. Strigin, R. Sturani, A. L. Stuver, T. Z. Summerscales, L. Sun, P. J. Sutton, B. L. Swinkels, M. J. Szczepańczyk, M. Tacca, D. Talukder, D. B. Tanner, M. Tápai, S. P. Tarabrin, A. Taracchini, R. Taylor, T. Theeg, M. P. Thirugnanasambandam, E. G. Thomas, M. Thomas, P. Thomas, K. A. Thorne, K. S. Thorne, E. Thrane, S. Tiwari, V. Tiwari, K. V. Tokmakov, C. Tomlinson, M. Tonelli, C. V. Torres, C. I. Torrie, D. Töyrä, F. Travasso, G. Traylor, D. Trifirò, M. C. Tringali, L. Trozzo, M. Tse, M. Turconi, D. Tuyenbayev, D. Ugolini, C. S. Unnikrishnan, A. L. Urban, S. A. Usman, H. Vahlbruch, G. Vajente, G. Valdes, M. Vallisneri, N. van Bakel, M. van Beuzekom, J. F. J. van den Brand, C. Van Den Broeck, D. C. Vander-Hyde, L. van der Schaaf, J. V. van Heijningen, A. A. van Veggel, M. Vardaro, S. Vass, M. Vasúth,

R. Vaulin, A. Vecchio, G. Vedovato, J. Veitch, P. J. Veitch, K. Venkateswara, D. Verkindt, F. Vetrano, A. Viceré, S. Vinciguerra, D. J. Vine, J.-Y. . Vinet, S. Vitale, T. Vo, H. Vocca, C. Vorvick, D. Voss, W. D. Vousden, S. P. Vyatchanin, A. R. Wade, L. E. Wade, M. Wade, S. J. Waldman, M. Walker, L. Wallace, S. Walsh, G. Wang, H. Wang, M. Wang, X. Wang, Y. Wang, H. Ward, R. L. Ward, J. Warner, M. Was, B. Weaver, L.-W. . Wei, M. Weinert, A. J. Weinstein, R. Weiss, T. Welborn, L. Wen, P. Weßels, T. Westphal, K. Wette, J. T. Whelan, S. E. Whitcomb, D. J. White, B. F. Whiting, K. Wiesner, C. Wilkinson, P. A. Willems, L. Williams, R. D. Williams, A. R. Williamson, J. L. Willis, B. Willke, M. H. Wimmer, L. Winkelmann, W. Winkler, C. C. Wipf, A. G. Wiseman, H. Wittel, G. Woan, J. Worden, J. L. Wright, G. Wu, J. Yablon, I. Yakushin, W. Yam, H. Yamamoto, C. C. Yancey, M. J. Yap, H. Yu, M. Yvert, A. Zadrożny, L. Zangrando, M. Zanolin, J.-P. . Zendri, M. Zevin, F. Zhang, L. Zhang, M. Zhang, Y. Zhang, C. Zhao, M. Zhou, Z. Zhou, X. J. Zhu, M. E. Zucker, S. E. Zuraw, and J. Zweizig. Observation of gravitational waves from a binary black hole merger. *Physical Review Letters*, 116(6), February 2016.

[2] 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.

[3] 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.

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

[5] 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.

[6] Susan M. Baxter, Steven W. Day, Jacquelyn S. Fetrow, and Stephanie J. Reisinger. Scientific software development is not an oxymoron. *PLoS Computational Biology*, 2(9):e87, September 2006.

[7] 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.

[8] 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.

[9] 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.

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

[11] 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.

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

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

[14] Landon Fuller, Kevin Van Vechten, and Jordan Hubbard. The MacPorts project – home, 2002.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

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

[20] 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.

[21] 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`, December 2014.

[22] 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.

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

[24] 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.

[25] 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.

[26] 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.

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

[28] 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.

[29] 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.

[30] 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.

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

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

[33] 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.

[34] S. Samadi, N. Almaeh, R. Wolfe, S. Olding, and D. 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.

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

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] 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.

[42] 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.

[43] Software discovery index. Available on the World Wide Web at http://softwarediscoveryindex.org/, 2014.

[44] 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.

[45] 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.

[46] 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.

[47] VA Software. SourceForge.net. Available on the World Wide Web at http://sourceforge.net, 1999.

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

[49] 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.

[50] 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.

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

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