The Rich and Fragmented Social Signals Developers Find for Reusing Open Source Libraries

Andrew Head
UC Berkeley
Berkeley, CA, USA
andrewhead@berkeley.edu

ABSTRACT

Programmers reuse software all the time. So much knowledge has migrated online, in the form of tutorials, questions and answers, and issues, authored by many and over the course of long times. We run a study to characterize the extent to which information about programming packages is fragmented online. We specifically focus on the problem of developers trying to learn about the health of the community and documentation for packages before using them. We report several interesting findings. For instance, **finding 1**. Also, **finding 2**.

CCS Concepts

•**Software and its engineering** → **Documentation;** *Search-based software engineering;* •**Human-centered computing** → *Collaborative and social computing systems and tools;*

Keywords

Something about search; information fragmentation

1. INTRODUCTION

Shorten to two paragraphs. Merge with background.

Software developers decide to reuse software to save time [5]. Researchers have observed that given a task, developers can assemble working programs through components found entirely on the web, together with their prior knowledge (e.g., [1]). This behavior is not uncommon—foraging for information online has been observed in particular for a variety of non-professional programmers [2] and validated through the queries for a help search engine for developer tools [1].

Though there are hazards in how developers select reusable components for software in practice. A premium is placed on working source code examples [6]. Developers' tend to choose code that "satisfices" without thoroughly testing it [1]. APIs have unexpected and undocumented side effects [7]. And despite a rise of "socially enabled" digital media for communicating about software, programmers face challenges using these channels: they may be overwhelmed by the amount of content and distrust its quality [8].

In this study, we present a view of how developers learn about packages on socially enabled channels. Through this study, we highlight two problems: First, programmers attempt to make sense of distributed information online. Second, some information is hidden from view, only to be discovered after tens of minutes of inspection. For us, these findings will motivate the design of new search systems for developers looking for resuable software components.

FSE Student Research Competition '16 November 13–18, 2016, Seattle, WA, USA
© 2016 ACM. ISBN 123-4567-24-567/08/06...$15.00
DOI: 10.475/123_4

2. BACKGROUND

Storey et al. [8], in their survey of developers' use of media channels, present a list of digital channels, and digital channels enabled with social media, that we find to be representative of the channels that developers use for support. In their survey, Storey et al. ask developers to name three channels that they find most important for development. In our analysis, we focus on some of the top digital channels that developers mentioned the most, including code hosting, Q&A, web search, and micro-blogging.

Developers face challenges when seeking support for software that they are reusing. The right information is often missing in the place that developers are looking for it [7, 8, 6]. The right components may be accessible only with terms or concepts the developer isn't familiar with [6, 3, 7] or require dependencies the developer doesn't know how to use. Developers can find documentation untrustworthy [7, 8], misleading, wrong, and out of date [6, 4, 7]. The speed of answers on mailing lists and Q&A sites can vary and take longer than developers want; slow response times can deter developers from asking questions or using a library. Furthermore, information can be organized in a way that developers find costly to navigate [7], and can be fragmented in a way that requires developers to look in multiple places to find the information they need [3].

There are already signals that programmers use to determine choices about trusting information and how to seek support online. Some developers may rely on experience and intuition [8]. They may inspect cosmetic features of web pages [1]. Authority and credibility of code examples can be assessed based on knowledge of and respect for the author of the code and evidence that the example is up to date [7]. In short, there is evidence that surface-level features help in the moment when deciding whether to make use of online documents, and an understanding of project authors' background and history can help a developer decide what components to use and when to ask questions.

3. APPROACH

Shorten to three paragraphs.

The objective for this study is to answer these questions. First, what channels are most helpful to developers predicting the quality of community and documentation available for packages? Second, what information and indications do they use from these channels to answer these questions? Third, what challenges do developers face when answering each question?

We designed a study to elicit answers to these questions. Participants were given the names of two Python packages that they were supposed to compare in terms of quality of community and documentation. These packages were chosen related to a task the participant expressed interest in, though we made sure the participant had never used either of the packages before.

We asked developers to attempt to answer six questions about the quality of the community and documentation:

1. Which package's **developers can you better trust** to make reliable, usable software?
2. Which community will be more **welcoming** when responding to questions you ask?
3. Which package will have **better How-To documentation** for all the tasks you will want to do?
4. Which package was better designed for users with **your technical knowledge** and goals?
5. For which package are other developers more likely to answer questions you ask as **fast** as you need them to?
6. Which package's documentation will be more **up-to-date** with the code?

These questions came from a variety of sources referenced in our background work and from our peers. We had a hunch that there might be some appropriate information sources online for answering each of these questions, but we didn't know for sure, or the variety of different approaches that developers might take. For each question, participants were given six minutes to use the web to learn more about the two packages they were comparing, in order to arrive at an informed answer. **Warmup task**.

We collected a variety of measures. We logged all URLs that participants visited along with timestamps of how long they spent on each site to determine how much time they were spending on each site to answer each question. We also requested that participants report the helpfulness of pages that they visited as "very helpful", "somewhat helpful", or "not helpful" for answering each question. (In practice, we had to remind them of this quite frequently, and received sparse ratings that did not cover many of the pages visited). We also asked participants to report their confidence in their comparison between the two packages, knowing that participants may only find a fraction of the available information on the web and in the time frame we gave them.

We also asked participants to produce two forms of written feedback. Before they began to search to answer this question, they were asked to share a 1–2 sentence strategy of how they expected to find an answer to the question. We expect this not only allowed participants to mentally prepare for the upcoming task, but also allowed us to ask after the study which of their strategies didn't succeed. After they searched, we asked them to report what evidence they found most helpful for comparing the two packages.

We have currently run the study with ten participants. **These participants have this background. . . .** We report the results in the next section.

We have made our Firefox addon for logging URL activity open source and available to the research community[1].

4. RESULTS

Shorten to four paragraphs.

Include the major insights that developers had during the study. For example: there's a new major version; there's an examples repository that wasn't clearly shown; there's a Google Groups page for the project; the docs are actually pretty usable. From this (and hopefully other insights) we see that a surface appearance of a package takes a while to build up accurately, and might require looking at a package from multiple different angles. We think that it need not take so much time.

Not all developers seemed to agree on what information sources were best. Part of this is likely because of prior knowledge. One of the participants chose to look at the IRC channels for the package.

I want to touch upon each of the following.

What questions are participants least confident answering?

What are the most authoratative sources for answering each question?

What do they approximate to aggregate?

How much do participants' perceptions of documentation change? And how much do their opinions change?

And what does this all suggest for what's broken for information presentation, and the design of programmers' front-end to the web?

5. CONCLUSIONS

In summary, we rock. We see challenge and solutions in helping developers better understand reusable components that they find online by capturing and summarizing fragmented information about its system image (i.e. documentation) and support history at the component level.

6. REFERENCES

[1] J. Brandt, P. J. Guo, J. Lewenstein, M. Dontcheva, and S. 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.

[2] J. Brandt, P. J. Guo, J. Lewenstein, and S. R. Klemmer. Opportunistic programming: How rapid ideation and prototyping occur in practice. In *Proceedings of the 4th International Workshop on End-user Software Engineering*, WEUSE '08, pages 1–5, New York, NY, USA, 2008. ACM.

[3] S. Y. Jeong, Y. Xie, J. Beaton, B. A. Myers, J. Stylos, R. Ehret, J. Karstens, A. Efeoglu, and D. K. Busse. Improving documentation for esoa apis through user studies. In *End-User Development*, pages 86–105. Springer, 2009.

[1]https://github.com/andrewhead/Web-Navigation-Logger

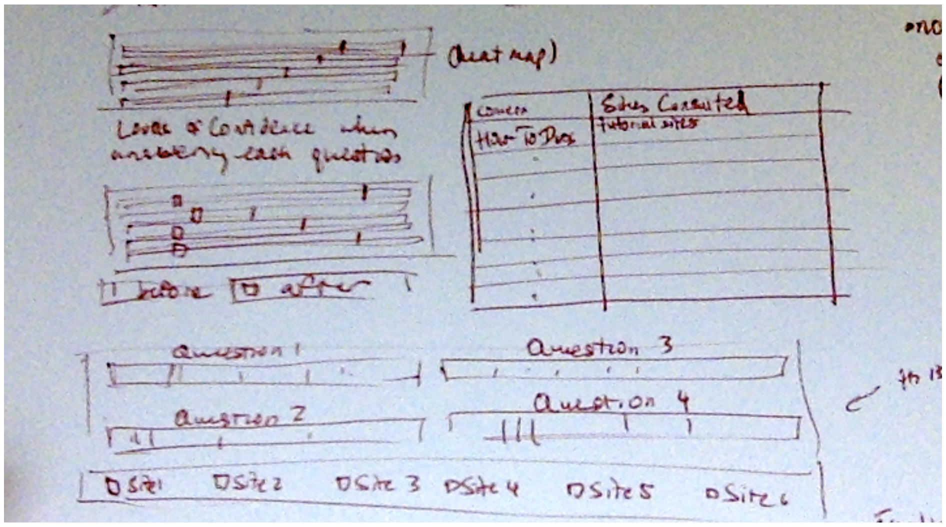

Figure 1: This is the figure that goes at the top of the page

[4] T. C. Lethbridge, J. Singer, and A. Forward. How software engineers use documentation: the state of the practice. *IEEE Software*, 20(6):35–39, Nov 2003.

[5] H. Mili, F. Mili, and A. Mili. Reusing software: issues and research directions. *IEEE Transactions on Software Engineering*, 21(6):528–562, Jun 1995.

[6] J. Nykaza, R. Messinger, F. Boehme, C. L. Norman, M. Mace, and M. Gordon. What programmers really want: Results of a needs assessment for sdk documentation. In *Proceedings of the 20th Annual International Conference on Computer Documentation*, SIGDOC '02, pages 133–141, New York, NY, USA, 2002. ACM.

[7] M. P. Robillard and R. Deline. A field study of api learning obstacles. *Empirical Software Engineering*, 16(6):703–732, 2011.

[8] M.-A. Storey, L. Singer, B. Cleary, F. Figueira Filho, and A. Zagalsky. The (r) evolution of social media in software engineering. In *Proceedings of the on Future of Software Engineering*, FOSE 2014, pages 100–116, New York, NY, USA, 2014. ACM.