A Study of Information Fragmentation on the Web for Programmers Foraging for Package Health

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

Research problem and motivation. Also, include background and related work.

Software developers often make the rational decision to reuse software to save time. This has been true for a long time [10]. 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., [4]). This behavior is not uncommon—foraging for information online has been observed in particular for a variety of non-professional programmers [5] and validated through the queries for a help search engine for developer tools [4].

The past five decades of software engineering have seen the rise of digital media for programmer communication and information sharing. As of the '90s, the web has "socially enabled" many new digital media, transforming how developers learn about new technologies [14]. In the last ten years, millions of projects migrated to GitHub, a social code hosting site, and hundreds of thousands of packages of reusable software have been published on globally available and instantly accessible "package indexes" for the Ruby, JavaScript, and Python languages. Not only has reusable software moved online en masse, but so too has much of the documentation for this work.

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

We see hazards in how developers select reusable components for software: A premium is placed on working source code examples [11]; developers' tend to choose code that "satisfices" without thoroughly testing it [4]; APIs have unexpected and undocumented side effects [13]; programmers don't necessarily know how to understand the digital media they use; they report being overwhelmed by the volume of content; and they can't trust the quality of content [14].

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. We believe that such representations, shown in the right place when developers are choosing new components, reduce the hazards of poor reuse choices when developers lack media literacy, and can help programmers understand what components are safe to trust.

In this paper, we describe how indications of content quality can be harvested on a package level from four top digital channels that developers use for development [14], three of which are socially enabled: code hosting, Q&A sites, web search, and micro-blogging. We focus specifically on reusable components at the resolution of packages, or self-contained libraries that can be installed from a package index and easily imported into one's code. We make this choice as packages have a system image that, although distributed and "fragmented," is accessible through thoughtful and systematized queries to the search interfaces and APIs of social media and web search.

Our primary contribution is that we build a system that creates summaries of package support quality by collecting fragmented information across digital channels and summarizes them in lightweight representations. We report how developers perceive the trustworthiness of software given these representations. We then suggest which channels appear to be the richest for conveying variation on the quality of reusable components to developers who have not seen them before. We present several example interfaces that show how these indications can be incorporated into modern interfaces where programmers seek help and may encounter the option to install packages.

2. BACKGROUND

Storey et al. [14], 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. Digital channels include web search, content recommenders, rich content, private discussions, discussion groups, public chat, and private chat. Digital channels that were socially enabled included feeds and blogs, news aggregators, social bookmarking, question & answer sites, professional networking sites, developer profile sites, social network sites, microblogs, code hosting sites, and project coordination tools. We do not report non-digital channels, as from our review, these don't seem to have the same hazards of fragmented or missing information and anti-social behavior that was reported as being problematic for digital channels.

In their survey, Storey et al. ask developers to name three channels that they find most important for development. We concentrate on four digital channels (three that are socially enabled) that developers mentioned the most: code hosting, Q&A, web search, and micro-blogging. The first three of these were explicitly mentioned in the three interviews we conducted with participants. We feel concentrating on these channels provides a diversity of channels, with different ways of communicating and finding information.

Developers face challenges when seeking support for software that they are reusing, and this has been documented for decades. The right information is often missing in the place that developers are looking for it [13, 14, 11]. The right components may be accessible only with terms or concepts the developer isn't familiar with [11, 7, 13] or require dependencies the developer doesn't know how to use. Developers can find documentation untrustworthy [13, 14], misleading, wrong, and out of date [11, 9, 13]. 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 [13], and can be fragmented in a way that requires developers to look in multiple places to find the information they need [7].

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 [14]. They may inspect cosmetic features of web pages [4]. 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 [13]. 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.

During exploratory literature reviews, we encountered research in computer-supported cooperative work that offered interfaces to help readers assess the quality of socially curated content. Examples of representations include: heat maps of readers' judgments of article credibility [12]; vignettes and text formatting revealing discussion and conflict around content snippets [15, 3]; visualizations of wiki contributions by user [1]; change request activity during development [2]; and groups of open source project commit messages, clustered by topic [6]. From this small and very non-random sample, we have anecdotal evidence that there is research interest in providing front ends to help consumers of social content share opinions about credibility, assess authorship, and to coordinate teamwork.

Implications of this paper for other types of decision-making where understanding support would be helpful. Perhaps choosing appliances from Amazon.com, choosing online services and apps to pay a subscription for, software applications, etc.

3. APPROACH

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

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].

To obtain a list of packages, I suggest fetching packages that participants are not certain to know (i.e. ones that aren't ubiquitous within the Node.js ecosystem), and that represent varying levels of popularity online. We make a heuristic for packages that have some documentation online. We look for the pattern `npm install <package_name>` in all Stack Overflow posts tagged "node.js" or "npm," and extract all distinct package names, counting how often that package name appears. We choose three packages from each of these percentile ranges: 5–10%, 10–30%, 30–100%. We note that this might confound the Stack Overflow condition, as all packages will have at least a trivial amount of documentation.

Some participants may already know about some of the packages and authors that we show them, if they have some level of community support and the developer has been working with Node.js for some time. We ask all participants to check boxes to report whether they have heard of and whether they have used each package. We can use this to exclude some ratings from our analysis. If they know of authors, this might not be problematic, as this could be indicative of a need to summarize packages in terms of those who created them.

3.1 Measures

- their confidence in their judgment
- whether a participant completes a task in less time than was available
- ratings on a Likert scale of each quality question

3.2 Analysis

We use the technique described in [8] to assess whether there is a difference between the distributions of confidence scores for both factors. According to the recommendations from[2], we use a Mann-Whitney test for post-hoc pairwise comparisons. In this way, we report which channels appear to be most capable of answering each question.

Confidence scores will also be useful in initially assessing whether there is any distribution for which confidence is more than positive. If this assumption holds, then we can assert that there indeed are some channels for which such quality assessments are possible.

We do a qualitative coding of participants' comments for evidence of the cues and challenges of using each channel to answer these question; we will not look for quantitative significance in comments.

3.3 Limitations

We report some biases in the study design. One of the reasons that we include web search is because we already have some code written to collect data about web search, and an on-going theme of our work is to understand how to help programmers search better.

The script that we used to generate forms for our study can be found online at http://tinyurl.com/support-views-study-script.

[1]https://github.com/andrewhead/Web-Navigation-Logger
[2]http://yatani.jp/teaching/doku.php?id=hcistats:kruskalwallis

4. RESULTS

Results and contribution. 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.

6. REFERENCES

[1] O. Arazy, E. Stroulia, S. Ruecker, C. Arias, C. Fiorentino, V. Ganev, and T. Yau. Recognizing contributions in wikis: Authorship categories, algorithms, and visualizations. *Journal of the American Society for Information Science and Technology*, 61(6):1166–1179, 2010.

[2] A. Begel, Y. P. Khoo, and T. Zimmermann. Codebook: discovering and exploiting relationships in software repositories. In *2010 ACM/IEEE 32nd International Conference on Software Engineering*, volume 1, pages 125–134, May 2010.

[3] E. Borra, E. Weltevrede, P. Ciuccarelli, A. Kaltenbrunner, D. Laniado, G. Magni, M. Mauri, R. Rogers, and T. Venturini. Societal controversies in wikipedia articles. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 193–196, New York, NY, USA, 2015. ACM.

[4] 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.

[5] 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.

[6] A. Hindle, M. W. Godfrey, and R. C. Holt. What's hot and what's not: Windowed developer topic analysis. In *Software Maintenance, 2009. ICSM 2009. IEEE International Conference on*, pages 339–348, Sept 2009.

[7] 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.

[8] M. C. Kaptein, C. Nass, and P. Markopoulos. Powerful and consistent analysis of likert-type ratingscales. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pages 2391–2394, New York, NY, USA, 2010. ACM.

[9] 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.

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

[11] 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.

[12] P. Pirolli, E. Wollny, and B. Suh. So you know you're getting the best possible information: A tool that increases wikipedia credibility. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, pages 1505–1508, New York, NY, USA, 2009. ACM.

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

[14] 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.

[15] W. B. Towne, A. Kittur, P. Kinnaird, and J. Herbsleb. Your process is showing: Controversy management and perceived quality in wikipedia. In *Proceedings of the 2013 Conference on Computer Supported Cooperative Work*, CSCW '13, pages 1059–1068, New York, NY, USA, 2013. ACM.