Discovering Signals and Sources on the Web for Socially-Enabled Software Project Telemetry

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

ABSTRACT

In this abstract, we present a vision of socially-enabled software telemetry tools to help developers and maintainers make sense of the social strengths and hazards of software projects. We describe a research agenda for collecting and exposing information about community, documentation, and developers. We provide preliminary results from a study with developers, where we offer insights into what signals answer important social questions about software projects, and where these signals can be found on the web.

CCS Concepts

•Information systems → *Social networks;* Web mining; Web log analysis; Internet communications tools; **•Software and its engineering** → *Documentation; Open source model;* Search-based software engineering;

Keywords

Socially-enabled software telemetry; open source

1. INTRODUCTION

Developers increasingly find themselves as part of a social, connected, online network. This has its benefits—many developers leverage social media and other online channels to learn about new technology, to stay informed, and to connect with other developers [10, 12]. In fact, today we see a proliferation of socially-enabled channels [11] through which developers answer each others questions [6], help each other overcome bugs and learn new tools [8], and share information in many different forms at many different speeds. The knowledge and conversations of programmers are increasingly distributed across the web.

About a decade ago, Johnson et al. [2] described how collecting and displaying software telemetry data could help developers make sense of the many streams of measurements one could collect about their software project and process, including build failures, crashes, and source code contributions. Across builds, coding, and runtime, Johnson et al. provided a set of sensors and the metrics they would collect. Continuous monitoring of diverse signals could enable developers to make better products, with the right interfaces. For similar reasons, dashboards [13] can help developers keep track of their teammates and checkpoints on software by better capturing thick incoming information in interfaces that integrate well into the development environment.

We propose Johnson et al.'s work is currently missing a category of sensors that is increasingly relevant to clients and maintainers of open source software: social information on the web. We concern ourselves with the study of social signals mineable from the web and salient to project clients and maintainers, and the design of interfaces that integrate into the development environment to help developers monitor their projects and choose the right packages. In this abstract, we focus on the first of these: a study about what information on the web helps developers answer questions about the social health of open source projects. We believe this is an important first step to discovering what sensors are the right ones for socially-enabled software telemetry.

There are several online tools we see as precursors to much more powerful socially-enabled telemetry systems. Awesome Python [1] generates comparisons of arbitrary pairs of Python packages [2], with popularity and activity metrics presumably based on download counts and commit activity. Ruby Toolbox[3] and the 'packagequality' widget[4] compute package quality ratings that draw on download counts, responses to issues, and GitHub stars. While these can be proxies for a package's quality, we believe that this isn't enough for understanding the social life of packages. When using packages, developers are concerned with the trustworthiness of developers [9], how up-to-date the documentation is [11, 7, 5, 9], and whether the community is anti-social [11].

As developers will often have to make trade-offs between these factors when choosing software, we are skeptical that anything short of carefully selected and arranged samples from socially-enabled channels can help a developer make a truly informed judgment about a package's community when evaluating software and, after on-boarding, choosing how to ask for help. This abstract presents our inquiry into what these samples from developers' socially-enabled networks and web pages should be.

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

[1]https://python.libhunt.com/
[2]https://python.libhunt.com/project/pygame/vs/panda3
[3]https://www.ruby-toolbox.com/
[4]http://packagequality.com/

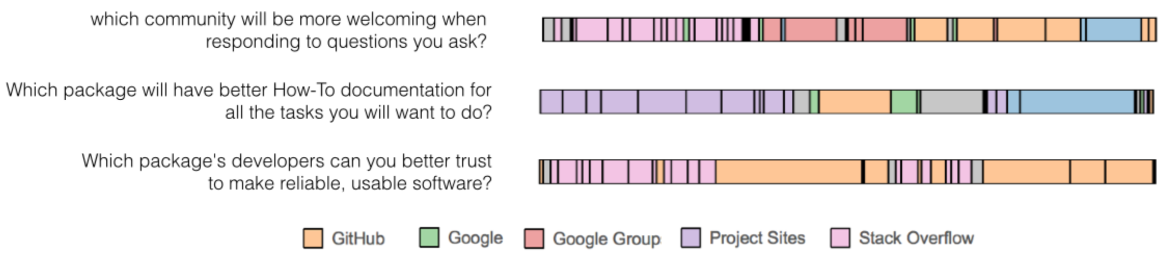

Figure 1: The sequence of communication channels one developer visited when answering questions that ask them to compare the quality of packages' community, documentation, and developers. Though we have found that users often rely on different channels, though this one relies on an interesting assortment: project documentation, GitHub, and blog posts to learn about the availale How-To documentation; Stack Overflow, Reddit, and Google Groups to learn about how welcoming the community is; GitHub profile pages, issues, pull request, and Stack Overflow to learn more about the developers behind the package.

2. OUR APPROACH

A space of tools for socially-enabled software telemetry has several dimensions. They can satisfy project clients seeking to learn more about projects, or project maintainers understand the conversations users are having, and the ones they could have with other projects. Telemetry systems can make use of signals, where these signals transmitted to users should convey understanding about one of at least three concerns: the community, the documentation, or the developers. For this work, we are concerned with the question: what signals can package clients use to answer questions about packages' communities, documentaiton, and developers, and where should they come from?

To answer this question, we present preliminary results from an in-progress study. We invited developers to partake in a 90-minute study where they were asked to evaluate two packages. We asked them to compare the packages with respect to their quality on six social dimensions related to community, documentation and developers, inspired from literature and conversations with software developers. We asked participants to use only the web to answer these questions. Evenn though developers often get their questions answered face-to-face [4, 11], over email [4, 3], or even by trial-and-error [1], we focus specifically on information that resides on the web, as we feel this will form the basis for mineable signlas for socially-enabled telemetry.

We collect three measurements of signals developers use to answer social questions: a timestamped log of URLs they visit; self-reported ratings of web pages' "helpfulness" for answering each question; open-ended responses of what evidence on the web was most helpful for comparing the social health of the two packages for each dimension. We report a cross-section of our results here.

3. PRELIMINARY RESULTS

Our data provides us with a rich picture of where developers find social signals. We show a portrait in Figure 1 for just one developer answering three of the six questions. Many domains that have been recently described as important to programmers—for instance, GitHub, Google, and Stack Overflow [11]—all have their place here, for various concerns. Developers offload questions about the recency of documentation to GitHub issues, commit histories, and pull request contents; they ask Stack Overflow, Reddit, and Google Groups about how welcoming communities are; and GitHub profiles help determine developer histories.

We want to highlight a few guidelines that we anticipate for the design of social signals, based on the strategies we have seen developers undertake when answering these questions about packages. First, references to text are necessary when developers are determining how welcoming communities are, and to understand whether their background matches that of the intended user. Second, developers delegate questions to Google, Reddit, Quora, and blog posts. On these channels, the presence or absence of information can be an answer in itself. Third, given the organization of the web, deverloper inevitably need to sample questions, documents, and issues to develop approximate answers to complex questions about packages.

Furthermore, these preliminary findings motivate the need for better client-facing socially-enabled software telemetry displays. It is common for a participant in our study to make a realization after twenty or thirty minutes of searching that changes their perception of a package, often only after the right question is asked—for example, discovering that a package's primary maintainer has been inactive for two years. Furthermore, whearas quickly skimming pages for a package may provide a surface-level understanding of it's quality, w ehave seen that serious, sustained attention to the right code and documentation can help developers overcome early biases and premature negative judgments. We believe that the obstacles developers face uncovering the true nature and strengths of packages is a product of developers' processes and the limitations of current information interfaces. It's only with attention to both of these that we will be able to design the first generation of telemetry systems for helping developers answer deep, complex, social questions about software projects.

4. CONCLUSIONS

In this extended abstract, we propose that new views are needed to help developers make sense of a sprawl of social information for the projects the evaluate, use, and maintain. By presenting preliminary results from a study with developers, we show evidence of what social signals exist on the web and where. This sets a research agenda for the design of socially-enabled software telemetry interfaces.

5. 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] P. Johnson, H. Kou, M. Paulding, Q. Zhang, A. Kagawa, and T. Yamashita. Improving software development management through software project telemetry. *IEEE Software*, 22(4):76–85, July 2005.

[3] A. J. Ko, R. DeLine, and G. Venolia. Information Needs in Collocated Software Development Teams. In *Proceedings of the 29th International Conference on Software Engineering*, ICSE '07, pages 344–353, Washington, DC, USA, 2007. IEEE Computer Society.

[4] T. D. LaToza, G. Venolia, and R. DeLine. Maintaining Mental Models: A Study of Developer Work Habits. In *Proceedings of the 28th International Conference on Software Engineering*, ICSE '06, pages 492–501, New York, NY, USA, 2006. ACM.

[5] 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.

[6] L. Mamykina, B. Manoim, M. Mittal, G. Hripcsak, and B. Hartmann. Design lessons from the fastest q&a site in the west. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 2857–2866, New York, NY, USA, 2011. ACM.

[7] 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.

[8] C. Parnin, C. Treude, and M. A. Storey. Blogging developer knowledge: Motivations, challenges, and future directions. In *2013 21st International Conference on Program Comprehension (ICPC)*, pages 211–214, May 2013.

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

[10] L. Singer, F. Figueira Filho, and M.-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.

[11] 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.

[12] M. A. Storey, A. Zagalsky, F. Filho, L. Singer, and D. German. How Social and Communication Channels Shape and Challenge a Participatory Culture in Software Development. *IEEE Transactions on Software Engineering*, PP(99):1–1, 2016.

[13] C. Treude and M.-A. Storey. Awareness 2.0: Staying Aware of Projects, Developers and Tasks Using Dashboards and Feeds. In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ICSE '10, pages 365–374, New York, NY, USA, 2010. ACM.