Post-literate Programming: Linking Discussion and Code in Software Development Teams

Post-literate Programming: Linking Discussion and Code in Software Development Teams

Soya Park, Amy X. Zhang, David R. Karger
MIT CSAIL
Cambridge, MA 02139, USA
[soya, axz, karger]@mit.edu

ABSTRACT

The *literate programming* paradigm presents a program interleaved with natural language text explaining the code's rationale and logic. While this is great for program *readers*, the labor of creating literate programs deters most program *authors* from providing this text at authoring time. Instead, as we determine through interviews, developers provide their design rationales after the fact, in discussions with collaborators. We propose to capture these discussions and incorporate them into the code. We have prototyped a tool to link online discussion of code directly to the code it discusses. Incorporating these discussions incrementally creates *post-literate programs* that convey information to future developers.

ACM Classification Keywords

H.5.2. Information Interfaces and Presentation (e.g. HCI): User Interfaces

Author Keywords

software development; group chat; knowledge management

INTRODUCTION

Managing knowledge and its spread within software development teams is challenging [16]. To streamline the knowledge sharing process, software teams rely on a suite of different tools, including multiple discussion channels such as email, online forums, community Q&A sites, and group chat, as well as project management tools and online code repositories. The multiplicity of tools can make it hard for developers to find the information they need or from which they could potentially benefit [10, 8]. Previous work suggests that documentation generation [12, 5, 2] and automatic commenting [11, 17] could be a way to share the design and structure of software implementations. Better tools for searching or inquiry within a code base [15, 14, 7, 13, 6, 1] have also been proposed to help developers locate or understand a piece of code. For instance, Codeon [3] introduces asynchronous assistance from remote developers to answer particular questions from developers.

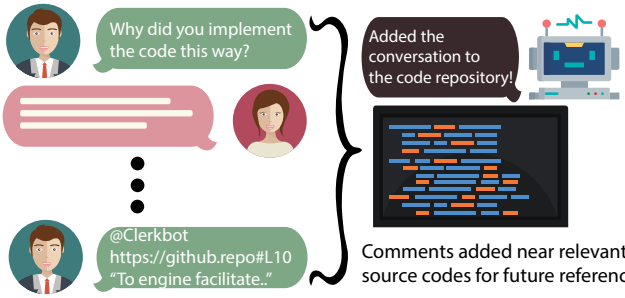

Figure 1. Workflow of Clerkbot: (a) developers converse about a section of code in chat, (b) a user alerts Clerkbot and specifies where the code exists as well as optionally provides a summary of the discussion, (c) the discussion summary and a link to the full discussion is added to the relevant source code as comments.

Such discussions generally take place on communication platforms separated from the code, which makes it hard to find them later (or even know they exist). In this work, we focus on preserving the information that is exchanged within online discussion platforms of software teams, and making it more easily accessible from code in order to provide context for programming. We call this incorporation of discussion about code within the code itself *post-literate programming*. In contrast to literate programming [9], where developers write their explanations about code while implementing the code itself, post-literate programming generates contextual information after the code has been written, by making use of discussions about the code.

We began by investigating how developers currently answer the question *"Why did my colleague implement the code like this?"*. We interviewed software developers from various sizes of teams and found that developers often spend time going from code to then searching through external resources. We also found that developers' most preferred way to answer the question is to use the revision history[1] in their version control system and then sometimes ask the author of the revision for details or pointers.

[1]https://git-scm.com/docs/git-blame

UIST '18 Adjunct October 14–17, 2018, Berlin, Germany

ACM ISBN 978-1-4503-5949-8/18/10…$15.00
DOI: https://doi.org/10.1145/3266037.3266098

From this finding, we built a discussion tool called ClerkBot that instantiates some of the ideas behind post-literate programming. ClerkBot is a connector between the discussion platform Slack and a code repository. Users can mention ClerkBot while chatting about code within Slack and the tool then contributes a commented line within the source code containing a link back to the discussion. The next time a developer is inspecting that piece of code or looking through the revision history, they can be directed to the discussion about the code for additional context. Future iterations of ClerkBot could link even more discussion platforms to further enrich the code representation.

NEEDFINDING INTERVIEWS

To investigate how developers access the deliberation history of code development, we conducted semi-structured interviews. We recruited 7 developers (2 females, 5 males, mean age=25.1). Interviewees were recruited through mailing lists and word-of-mouth. Four of the interviewees are working on research projects or course assignment, and the rest are working on commercial products. We had a one-on-one in-person interview for 1 hour for each participant. The participants were each compensated $20 for their time.

In each interview, we asked about team dynamics and the interaction between team members such as their communication tools and team meeting schedule. We also asked them to bring to the interview one or two discussions within their team and annotate what content interviewees found useful and how they revisited that content. The following summarizes comments from the interviews.

Developers share knowledge through discussion and often go back to old discussions: If developers know whom to ask questions, they ping their colleague; otherwise, they post in the group discussion. Interviewees said they often don't read discussions in real-time but instead mark them to revisit later by starring or forwarding to themselves. One person said: *"I set aside time to comprehend the team discussion. I normally star the discussion and get back to it later unless I'm mentioned in the discussion."*

Developers often inspect the source code by tracking revision history to understand the implementation: Since software development teams rely on multiple channels of information, developers are often overwhelmed when trying to find information about code. However, interviewees said tracking the revision history often guides them in the right direction to begin searching.

CLERKBOT: LINKING DISCUSSION AND CODE

Clerkbot is a chatbot tool that developers in a team can use to mark conversations that are then linked to their code repository. Following are design considerations for Clerkbot.

Link implementation and discussion closer: Software teams use many different tools in order to organize different types of tasks into different workspaces. However, this dispersion leads to information in different tools being disconnected. From our interviews, we saw that developers don't update comments because editing comments requires many tedious steps (e.g. commit, pull request), even though information gleaned from discussions can be helpful for understanding the code base.

Scale and Scope of Comments: An open design question is how much of a given discussion should be incorporated in code comments. Providing only a link to the discussion minimizes code clutter but also requires the reader to navigate elsewhere to learn about it. At the other extreme, incorporating the entire discussion provides full information but may overclutter the code. As a middle ground, we could incorporate only a summary, with a link to the full discussion. As one interviewee said, *"I don't want to see the entire thread.. I only want a short summary of a final design decision and maybe second and third alternatives."* Thus, Clerkbot can add a summary of the discussion to the commented line, along with the link to the full discussion. However, summarizing discussions require extra effort from developers. As a future work, we will design an emergent summarization tool to ease the summarization task [18].

The scope of a discussion can also range widely from one line of source code to functions to general design decisions of software. Clerkbot should locate any discussion at a suitable scope and location of the source code, so that future developers can find the discussion at the expected place.

Avoiding Comment Clutter: To avoid cluttering the code base with too many comments attached by Clerkbot, Clerkbot can show nearby comments already inserted by Clerkbot and suggest to users to combine their comment with another comment. In this way, users can manage to avoid redundant content and not clutter the code with different comments.

FUTURE WORK

Clerkbot suggests a method of preserving valuable knowledge scattered in various communication channels. However, this work raises additional future work to enhance knowledge sharing experiences within code:

IDE plug-in: To address concerns about cluttering code with too much discussion, an IDE plug-in that can present discussions *beside* the code would reduce clutter *in* the code—a form of code annotation. Such an IDE would allow users to view full discussions without leaving their IDE.

Presentations of large-scale knowledge sharing: The current implementation of Clerkbot only allows a linear view of raw discussions. However, the linear view is hard for navigation or comprehension if the amount of information piles up over time. As future work, we will investigate better presentations of knowledge sharing at scale, including hierarchical or alternate representations [19, 20].

Social factors to recommend relevant knowledge: These tools could also encapsulate social dynamics [4] in discussion tools of software development teams to automatically detect the relevance of information for each developer and recommend if the information is of interest (e.g. working on a similar issue, an important issue to the team, interesting to colleagues) to the developer.

REFERENCES

1. Sushil Bajracharya, Trung Ngo, Erik Linstead, Yimeng Dou, Paul Rigor, Pierre Baldi, and Cristina Lopes. 2006. Sourcerer: a search engine for open source code supporting structure-based search. In *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*. ACM, 681–682.
2. Denise Che. 2014. *Automatic documentation generation from source code*. Ph.D. Dissertation. Massachusetts Institute of Technology, Cambridge, MA.
3. Yan Chen, Sang Won Lee, Yin Xie, YiWei Yang, Walter S. Lasecki, and Steve Oney. 2017. Codeon: On-demand software development assistance. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. ACM, 6220–6231.
4. L. Dabbish, C. Stuart, J. Tsay, and J Herbsleb. 2012. Social coding in GitHub: transparency and collaboration in an open software repository. In *Proceedings of the ACM 2012 conference on computer supported cooperative work*. ACM, 1277–1286.
5. Uri Dekel and James D. Herbsleb. 2009. Improving API documentation usability with knowledge pushing. In *Proceedings of the 31st International Conference on Software Engineering*. IEEE Computer Society, 320–330.
6. Brian Eddy. 2014. Structured source retrieval for improving software search during program comprehension tasks. In *Proceedings of the companion publication of the 2014 ACM SIGPLAN conference on Systems, Programming, and Applications: Software for Humanity*. ACM, 13–15.
7. Emily Hill, Lori Pollock, and K. Vijay-Shanker. 2009. Automatically capturing source code context of nl-queries for software maintenance and reuse. In *Proceedings of the 31st International Conference on Software Engineering*. IEEE Computer Society, 232–242.
8. Peter Loffler Jarczyk, Alex PJ and Frank M. Shipmann. 1992. Design rationale for software engineering: a survey. In *Proceedings of the Twenty-Fifth Hawaii International Conference*. IEEE, 577–586.
9. Donald Ervin Knuth. 1984. Literate programming. *Comput. J.* 17, 2 (1984), 97–111.
10. Andrew J. Ko, Robert DeLine, and Gina Venolia. 2007. Information Needs in Collocated Software Development Teams. In *Proceedings of the 29th international conference on Software Engineering*. IEEE Computer Society, 344–353.
11. Mario Linares-VÃąsquez, Luis Fernando CortÃl's-Coy, Jairo Aponte, and Denys Poshyvanyk. 2015. Changescribe: A tool for automatically generating commit messages. In *Proceedings of the 37th International Conference on Software Engineering*. IEEE Press, 709–712.
12. Paul W. McBurney and Collin McMillan. 2014. Automatic documentation generation via source code summarization of method context. In *Proceedings of the 22nd International Conference on Program Comprehension*. ACM, 279–290.
13. Liming Nie, He Jiang, Zhilei Ren, Zeyi Sun, and Xiaochen Li. 2016. Query Expansion Based on Crowd Knowledge for Code Search. *IEEE Trans. Services Computing* 9, 5 (2016), 771–783.
14. Santanu Paul and Atul Prakash. 1994. A framework for source code search using program patterns. *IEEE Transactions on Software Engineering* 20, 6 (1994), 463–475.
15. Steven P Reiss. 2009. Semantics-based code search. In *Proceedings of the 31st International Conference on Software Engineering*. IEEE Computer Society, 243–253.
16. Ioana Rus, Mikael Lindvall, and S. Sinha. 2002. Knowledge management in software engineering. *IEEE software* 19, 3 (2002), 26–38.
17. Emily Hill Divya Muppaneni Lori Pollock Sridhara, Giriprasad and K. Vijay-Shanker. 2014. Towards automatically generating summary comments for java methods. In *Proceedings of the IEEE/ACM international conference on Automated software engineering*. ACM, 43–52.
18. Amy X. Zhang and Justin Cranshaw. 2018. Making sense of group chat through collaborative tagging and summarization. In *Proceedings of the ACM Conference on Computer Supported Cooperative Work and Social Computing (CSCW)*. ACM.
19. Amy X. Zhang, Lea Verou, and David Karger. 2017. Wikum: Bridging discussion forums and wikis using recursive summarization. In *Proceedings of the ACM 2012 conference on computer supported cooperative work*. ACM, 2082–2096.
20. Joyce Zhu, Jeremy Warner, Mitchell Gordon, Jeffery White, Renan Zanelatto, and Philip J. Guo. 2015. Toward a Domain-Specific Visual Discussion Forum for Learning Computer Programming: An Empirical Study of a Popular MOOC Forum. *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (2015), 101–109.