API Usage in Descriptions of Source Code Functionality

Paige Rodeghero, Collin McMillan, and Abigail Shirey
Department of Computer Science and Engineering
University of Notre Dame
Notre Dame, IN, USA
{prodeghe, cmc, ashirey}@nd.edu

***Abstract*—In this paper, we present a study exploring the use of API keywords within method summaries. We conducted a web-based study where we asked participants to rank Java method summaries based on five levels of detail, from low level to high level. We found that programmers widely use API in both high and low level summaries. Specifically, we found that 76.78% of higher level summaries contain Java API keywords. Additionally, we found that 93.75% of lower level summaries also contain them. This also shows that, in general, as the detail level decreases, the number of API keywords within the summary increases. It is our hope that this line of research will spark a discussion about API usage outside of source code. It is possible that method summaries are not the only form of documentation that API usage plays an important role. We believe these may be important results that could lead to an improvement for API usability design.**

I. Introduction

This paper explores the question of whether programmers use APIs to describe functionality implemented in source code that calls those APIs. The term "API usage" is usually reserved for occurrences in source code of calls to functions in the API, or reads and writes to data owned by an API. However, APIs have been shown to be key anchors for understanding source code [1], and it is plausible that programmers use APIs to describe source code in addition to using the APIs to implement features in their code. Given the strong social aspect to software development [2] and the productivity benefits of good communication tools [3], it is helpful to know if API usage extends beyond source code and into the dialog that programmers use to communicate.

We found evidence that programmers use APIs to describe code, even in short, high level summaries of code functionality. The implication is that APIs play a role in programmer communication beyond low level implementation details of the code itself. A mismatch has long been recognized between high level concepts used to describe software functionality, and the low level implementation details [4]. And while precise definitions of what constitutes "high" and "low" level in the literature are scant, it can be said that, generally speaking, a description at a high level includes information about the rationale behind the implementation, or the purpose of the software. In contrast, a description at a low level includes specifics about how the implementation was achieved, such as which methods occur in which class. Programmers use both high and low level descriptions to explain and understand source code, such as through documentation. Our hope for this paper is to generate discussion at the workshop about the role of APIs in communication.

To collect the evidence as a starting point for discussion, we used a publicly-available dataset of English summaries of functions in a program's source code [5]. That dataset was created by hiring professional programmers to read functions and write an English summary of those functions, as though their goal was to describe the function to another programmer. In this paper, we analyzed those summaries in two ways. First, we hired programmers to read the summaries and place them subjectively on a scale from low to high level. 577 were rated as high or leaning high level, while 496 were rated as low or leaning low level. Second, we searched each summary for function names from the standard Java API (after filtering out names that are also common English words). We found that 76.78% of high and leaning high level summaries contained references to the API, compared to 93.75% of low and leaning low level summaries. We also found that as the detail level lowers, the number of API increases. In short, we found evidence that programmers referred to the API in both high and low level summary descriptions. We have released all data used in this paper in an online appendix to facilitate discussion (see Section IV-E).

II. The Problem

The problem we target is that software engineering literature does not explore whether "API usage" extends to programmer conversations and other artifacts such as documentation, rather than solely usage in a program's source code. Software engineering efforts are spread across numerous artifacts, with source code being only one of several important components. APIs are generally considered to be critical to source code development, but the degree to which they affect other artifacts has not been explored deeply. Improved knowledge in this area could lead to advancements in tool support for programmers and API usability design. Possible workshop discussion points include: 1) perceptions of the value of API usage for non-source-code artifacts, 2) implications for existing tool support, 3) improved mechanisms to detect API usage in natural language, and 4) improved detection of the level of natural language summaries as pertaining to APIs.

III. Background and Related Work

In this section, we cover background information on API usage, method summaries, and our previous work that we extend upon in this paper.

A. API Usage

API documentation contains much information for many audiences [6]. Application developers, platform licensees, and conformance test engineers all use the Java API specification documentation. Developers might require elaborate descriptions than concise documentation comments can provide. However, including API specification in documentation comments helps both writers of conforming implementations and conformance tests maintain cross-platform functionality [7]. API directives tell developers how to implement the API by providing guidelines and constraints. Monperrus *et al.* [6] classified Java directives to find that method call directives were the most abundant at 43%.

McBurney *et al.* [8] studied the similarity in source code to summaries written by authors and readers. Reader summaries contained more keywords from the code and were more low-level summaries. Rastkar *et al.* [9] examined how to best summarize bug reports. Rastkar focuses on reports written as conversations, instead of reports containing mostly code and stack traces, as developers usually do not read low-level bug reports.

B. Method Summaries

Software engineers use many different strategies to understand source code [10]. Software documentation has long been recommended to summarize source code in a readable format [11]. Some believe that the most effective method summaries contain both detailed source code information and behavioral information, as well as its context [8]. Some others believe that a concise summary of the code is best for saving the user time and effort during comprehension. [12]. Ko *et al.* [13] investigated how software developers comprehend unfamiliar code without documentation. He found that on average they spent 35% of their time determining where the parts of the program were located and how they interacted. He concluded that environments to help developers access relevant information would help them work more efficiently and effectively.

Forward *et al.* [14] surveyed 48 people in industry to find documentation to be an important communication tool and should ideally be up-to-date. Automatic method summaries ensure that the documentation is consistent with its source code. Sridhara *et al.* [15] developed a summary comment generator that puts components of important code into natural-language templates. Novick *et al.* [16] surveyed 25 general computer users and discovered that the majority preferred documentation that is easy to navigate, technically appropriate, and had problem-oriented organization. Software engineers are more technically skilled, so above all they need up-to-date documentation to maintain systems with ease [17], or else they must turn to the source code [18].

C. Our Previous Work

We conducted three separate previous studies where we asked programmers to write method summaries based on Java code. The first study was an eye tracking study where we examined the keywords programmers fixated on, where the eyes focused for an extended amount of time, while reading Java methods [5]. The participants were asked to read, one method at a time, and summarize the method. This required the participant to comprehend the source code while we tracked their eyes. The second study had users comparing manual summaries against automatic summaries [19]. It wanted to see if the user study could help create an improved technique for summarization to include why the method exists in its context. The third study we conducted had programmers reading Java methods again, this time with obfuscated words first, and then the same method with all of the terms included [20]. We wanted to see if the programmers could comprehend the source code without all of the terms being included. We asked the participants to write a summary for the obfuscated words first, and then they were able to update or change their summary after seeing the entire method.

IV. Quantitative Study Design

In this section, we describe our research questions, the methodology of our study, and discuss the threats to validity.

A. Research Objective

The goal of this study is to find how much API keywords are used in different levels of summaries. Here, API keywords represent terms contained within the Java standard libraries, excluding any common English terms. Towards this goal, we propose two Research Questions (RQ):

RQ_1 Do summaries of all detail levels contain API keywords?
RQ_2 Do low level summaries contain more API keywords than high level summaries?

The rationale for RQ_1 is to determine if API usage plays an important role in method summaries. The rationale for RQ_2 is that we believe programmers may need various detail levels of auto-generated summaries. The results of this study could improve our knowledge of method summaries and API usage within them. Also, we could discover a potential factor that determines the difference between high level and low level summaries.

B. Methodology

Our methodology was to conduct a web-based study where we asked participants to rank Java method summaries based on the level of detail. We used data from a previous eye tracking study and previous method summary studies (see Section III-C). We randomized and selected from the 481 method summaries written by professional programmers from various companies such as IBM and Uber from the previous studies. We designed the web-based study to last ten minutes in length. The participants were shown one method summary at a time and asked to rank the summary displayed as High Level

(concise), Leaning High Level, Neither High nor Low Level (neutral), Leaning Low Level, or Low Level (exhaustive). Note: since the summaries were randomly selected to be shown to each participant, we cannot guarantee how many duplicates of each summary were ranked.

The following are examples of a High Level and Low Level summary, respectively. Both of the below summaries are about the same Java method.

High Level: *Returns the passed object as JSON.*

Low Level: *This method takes an object and wraps it, returning the JSON the object has become. If the object is already a JSON, it casts the object as a JSON and returns it.*

After the web-based study concluded, we took the participants' detail level rankings and checked to see how many Java API keywords were included in each ranking. We filtered out common English words from this search such as "append" and "remove". We specifically filtered for all the words included in the New General Service List (NGSL) [21]. The NGSL is a corpus of 2800 high frequency English words composed for the purpose of learning English.

C. Participants

The participants were students from the University of Notre Dame's Department of Computer Science and Engineering. There were a total of 27 participants. Of the 27 students, 9 of the of the participants were undergraduate students and 18 of the participants were graduate students. 7 of the students who participated were female. The participants' overall software experience ranged from 2.5 to 20 years. The participants had experience in many programming languages, including Java.

D. Threats to Validity

This quantitative study has four main threats to validity. The first threat is that the participants' selection of the levels of detail could be a bit subjective. We attempted to mitigate this threat by having a wide variety of programmers participate. The second threat is that the programmers who wrote the summaries may have not necessarily written helpful or correct descriptions of the Java methods. However, we believe this was reduced by using professional programmers to write the summaries. The third threat is that each participant had only ten minutes to complete the study. However, we believe plenty of rankings were collected during that time. Also, some participants were able to complete more ratings than others. Finally, the fourth threat is the varying level of experience throughout the participants. This was also mitigated by using a large enough number of participants to reduce outlying results.

E. Reproducibility

All our input data, raw and processed results, and processing scripts are available via our online appendix [1] for use by other researchers.

[1] http://www3.nd.edu/~prodeghe/projects/highlow/

V. Quantitative Study Results

From the analysis of the summaries, we found that references to Java API were widely found in all detail levels of summaries. We also found that lower levels of detail tended to have a higher rate of API usage than higher levels.

A. RQ_1: API Usage

1) High Level: Participants labeled 324 summaries as high level. Of those ranked high, 226 contained Java API usage, meaning API appeared in about 69.75% of all high level summaries.

2) Leaning High Level: Participants labeled 253 summaries as leaning high level. Of those ranked leaning high, 217 contained Java API usage, meaning API appeared in about 85.77% of all leaning high level summaries.

3) Neutral Level: Participants labeled 184 summaries as neither high or low level. Of those ranked neutral, 151 contained Java API usage, meaning API appeared in about 82.07% of all neutral level summaries.

4) Leaning Low Level: Participants labeled 285 summaries as leaning low level. Of those ranked leaning low, 272 contained Java API usage, meaning API appeared in about 95.44% of all leaning low level summaries.

5) Low Level: Participants labeled 211 summaries as low level. Of those ranked low, 193 contained Java API usage, meaning API appeared in about 91.47% of all low level summaries.

These results show that API usage can be seen throughout all summary levels.

B. RQ_2: High vs Low Level

Looking at the combined results of the two high detail levels, 577 summaries were rated as either high or leaning high level. Of these, 76.78% of them contained API. On the other hand, 496 summaries were rated as either low or leaning low level. These summaries included 93.75% of API. This more clearly shows the increased use of Java API in lower level summaries over higher level ones.

VI. Discussion

Our paper contributes to research on API documentation in two ways. First, we believe that our results show how APIs are mentioned in method summaries and how the use of them can provide more detailed summaries. Second, we also believe that if one wants to explicitly create a lower level summary, it should include more API usage. APIs are generally considered to be important for source code development, but how much they affect other artifacts has not been explored deeply. Increased knowledge in this area could lead to many useful improvements in tools for programmers. Our goal for the workshop is to receive community feedback on this line of research which could include broad implications for existing tool support, improved detection of API usage, and improved auto-generation for method summaries.

VII. Conclusion

We have presented a quantitative study about the amount of API usage in and difference between high level and low level manually written Java method summaries. We explore two research question aimed at understanding this existence of API keywords in these summaries. Through an web-based human study we were able to collect data on what programmers perceived as five different detail levels of method summaries. We showed that all detail levels contain API usage and that the lowest detail level of summaries *do* have more API keywords than higher level summaries. Our findings lead us to believe that a key distinction between a high level and low level method summary is the number of API keywords that are present.

VIII. Acknowledgments

We thank the undergraduate and graduate students from the University of Notre Dame's Department of Computer Science and Engineering for their participation in the study. This work was partially supported by the NSF grants CC-1452959 and DGE-1313583. Any opinions, findings, and conclusions expressed herein are the authors' and do not necessarily reflect those of the sponsors.

References

[1] C. McMillan, M. Grechanik, and D. Poshyvanyk, "Detecting similar software applications," in *Software Engineering (ICSE), 2012 34th International Conference on*. IEEE, 2012, pp. 364–374.

[2] W. Scacchi, "Managing software engineering projects: A social analysis," *IEEE Transactions on Software Engineering*, pp. 49–59, 1984.

[3] M.-A. Storey, C. Treude, A. van Deursen, and L.-T. Cheng, "The impact of social media on software engineering practices and tools," in *Proceedings of the FSE/SDP workshop on Future of software engineering research*. ACM, 2010, pp. 359–364.

[4] T. J. Biggerstaff, B. G. Mitbander, and D. E. Webster, "Program understanding and the concept assignment problem," *Communications of the ACM*, vol. 37, no. 5, pp. 72–82, 1994.

[5] P. Rodeghero, C. Liu, P. W. McBurney, and C. McMillan, "An eye-tracking study of java programmers and application to source code summarization," *IEEE Transactions on Software Engineering*, vol. 41, no. 11, pp. 1038–1054, Nov 2015.

[6] M. Monperrus, M. Eichberg, E. Tekes, and M. Mezini, "What should developers be aware of? an empirical study on the directives of api documentation," *Empirical Softw. Engg.*, vol. 17, no. 6, pp. 703–737, Dec. 2012. [Online]. Available: http://dx.doi.org/10.1007/s10664-011-9186-4

[7] D. Kramer, "Api documentation from source code comments: A case study of javadoc," in *Proceedings of the 17th Annual International Conference on Computer Documentation*, ser. SIGDOC '99. New York, NY, USA: ACM, 1999, pp. 147–153. [Online]. Available: http://doi.acm.org/10.1145/318372.318577

[8] P. W. McBurney and C. McMillan, "Automatic source code summarization of context for java methods," in *IEEE Transactions on Software Engineering*, vol. 42, Feb 2016, pp. 103–119.

[9] S. Rastkar, G. C. Murphy, and G. Murray, "Summarizing software artifacts: A case study of bug reports," in *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 1*, ser. ICSE '10. New York, NY, USA: ACM, 2010, pp. 505–514. [Online]. Available: http://doi.acm.org/10.1145/1806799.1806872

[10] T. Roehm, R. Tiarks, R. Koschke, and W. Maalej, "How do professional developers comprehend software?" in *Proceedings of the 34th International Conference on Software Engineering*, ser. ICSE '12. Piscataway, NJ, USA: IEEE Press, 2012, pp. 255–265.

[11] J. Greenberg and R. A. Baron, *Behavior in organizations: Understanding and managing the human side of work*. Pearson College Division, 2003.

[12] G. Sridhara, L. Pollock, and K. Vijay-Shanker, "Automatically detecting and describing high level actions within methods," in *Proceedings of the 33rd International Conference on Software Engineering*, ser. ICSE '11, New York, NY, USA, 2011, pp. 101–110. [Online]. Available: http://doi.acm.org/10.1145/1985793.1985808

[13] A. J. Ko, B. A. Myers, M. J. Coblenz, and H. H. Aung, "An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks," *IEEE Transactions on Software Engineering*, vol. 32, no. 12, pp. 971–987, Dec 2006.

[14] A. Forward and T. C. Lethbridge, "The relevance of software documentation, tools and technologies: A survey," in *Proceedings of the 2002 ACM Symposium on Document Engineering*, ser. DocEng '02. New York, NY, USA: ACM, 2002, pp. 26–33. [Online]. Available: http://doi.acm.org/10.1145/585058.585065

[15] G. Sridhara, E. Hill, D. Muppaneni, L. Pollock, and K. Vijay-Shanker, "Towards automatically generating summary comments for java methods," in *Proceedings of the IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '10. New York, NY, USA: ACM, 2010, pp. 43–52. [Online]. Available: http://doi.acm.org/10.1145/1858996.1859006

[16] D. G. Novick and K. Ward, "What users say they want in documentation," in *Proceedings of the 24th Annual ACM International Conference on Design of Communication*, ser. SIGDOC '06. New York, NY, USA: ACM, 2006, pp. 84–91. [Online]. Available: http://doi.acm.org/10.1145/1166324.1166346

[17] S. C. B. de Souza, N. Anquetil, and K. M. de Oliveira, "A study of the documentation essential to software maintenance," in *Proceedings of the 23rd Annual International Conference on Design of Communication: Documenting &Amp; Designing for Pervasive Information*, ser. SIGDOC '05. New York, NY, USA: ACM, 2005, pp. 68–75. [Online]. Available: http://doi.acm.org/10.1145/1085313.1085331

[18] B. Thomas and S. Tilley, "Documentation for software engineers: What is needed to aid system understanding?" in *Proceedings of the 19th Annual International Conference on Computer Documentation*, ser. SIGDOC '01. New York, NY, USA: ACM, 2001, pp. 235–236. [Online]. Available: http://doi.acm.org/10.1145/501516.501570

[19] P. W. McBurney and C. McMillan, "Automatic documentation generation via source code summarization of method context," in *Proceedings of the 22nd International Conference on Program Comprehension*. ACM, 2014, pp. 279–290.

[20] P. Rodeghero, "Discovering important source code terms," in *Proceedings of the 38th International Conference on Software Engineering Companion*, ser. ICSE '16. New York, NY, USA: ACM, 2016, pp. 671–673. [Online]. Available: http://doi.acm.org/10.1145/2889160.2891037

[21] C. Browne, "New General Service List," http://www.newgeneralservicelist.org, accessed: 01-23-2017.