# What Information about Code Snippets Is Available in Different Software-Related Documents? An Exploratory Study

Preetha Chatterjee*, Manziba Akanda Nishi†, Kostadin Damevski†,
Vinay Augustine‡, Lori Pollock* and Nicholas A. Kraft‡
* University of Delaware, Newark, DE, USA
Email: {preethac, pollock}@udel.edu
† Virginia Commonwealth University, Richmond, VA, USA
Email: {nishima, kdamevski}@vcu.edu
‡ABB Corporate Research, Raleigh, NC, USA
Email: {vinay.augustine, nicholas.a.kraft}@us.abb.com

***Abstract*—A large corpora of software-related documents is available on the Web, and these documents offer the unique opportunity to learn from what developers are saying or asking about the code snippets that they are discussing. For example, the natural language in a bug report provides information about what is not functioning properly in a particular code snippet. Previous research has mined information about code snippets from bug reports, emails, and Q&A forums. This paper describes an exploratory study into the kinds of information that is embedded in different software-related documents. The goal of the study is to gain insight into the potential value and difficulty of mining the natural language text associated with the code snippets found in a variety of software-related documents, including blog posts, API documentation, code reviews, and public chats.**

## I. Introduction

Integrated development environments today include sophisticated program modeling and analyses behind the scenes to support the developer in navigating, understanding, and modifying their code. While much can be learned from results of static and dynamic analysis of their source code, developers also look to others for advice and learning. As software development teams are more globally distributed and the open source community has grown, developers rely increasingly on written documents for help they might have previously obtained through in-person conversations.

Developer communications (e.g., developer blog posts, bug reports, API documentation, mailing lists, code reviews, and question & answer forums) offer the unique opportunity to learn from what the developers are saying or asking about the code snippets they are discussing. E-books, online course materials, videos, and programming manuals offer learning opportunities, often explaining concepts using code examples. Similarly, technical presentations and research papers often use code examples to demonstrate challenges addressed by their work. Developers can also learn from code in benchmark suites, standards documents, and code snippets from repositories such as GitHub Gists and Pastebin.

While individual developers can learn from the aforementioned sources, the publicly available, large corpora of software-related documents also provide the opportunity to automatically learn common API behaviors, properties, and vulnerabilities, as well as to complement similar information learned by mining software repositories. Researchers have performed empirical analysis of developer communications in order to understand specific software enclaves or novel techniques used by developers in the field, for instance, examining modern code reviews using tools like Gerrit [1], or the information encoded in tutorials for mobile development [2]. The captured insights can be integrated into the development of new models and analyses to enhance software engineering techniques and tools.

The knowledge that can be gained from these sources is typically embedded in the natural language describing the code or the properties it exhibits. For example, the natural language might note that the code is a good example of a correct, efficient, or secure solution to a particular problem, or that the code exemplifies a common code bug, security vulnerability, or energy bug to avoid, possibly in a particular context.

Researchers have also developed analyses to mine various information from the natural language text surrounding code snippets in emails [3], [4], bug reports [3], Q & A forums [5], [6], [7], and tutorials [8]. For example, Panichella et al. [3] developed a feature-based approach to automatically extract method descriptions from bug tracking systems and mailing lists. Vassallo et al. [5] built on their previous work [3] to design a tool called CODES that extracts candidate method documentation from StackOverflow discussions and creates Javadoc descriptions. Wong et al. [6] mine comments from Stack Overflow by extracting text preceding code snippets and using the code-description mappings in the posts to automatically generate descriptive comments for similar code snippets matched in open-source projects. Collectively, these efforts demonstrate the potential for extracting and using information embedded in natural language text of software-

related documents for software engineering tools. They also raise the questions of what other kinds of information can be extracted from these particular types of documents and of what kinds of information can be extracted from other types of software-related documents.

This paper describes an exploratory study that investigates the following questions:

1) What are the characteristics of the code snippets embedded in different types of software-related documents?
2) What kind of information is available in each document type? Which document types provide similar information? Which information is most readily available across all document types?
3) What are the cues that indicate code snippet related information? How do the cues differ across different document types? What influential words/phrases indicate the different kinds of information in different document types?

Our results provide preliminary indications regarding the document types that would serve as the most fruitful sources for different kinds of information, and also provide insights into what cues might help in automatically extracting that information.

## II. Methodology

Figure 1 depicts the major steps of our exploratory study. After selecting a set of documents of each document type under investigation, we redacted the code snippets, and asked human annotators to make as many observations about what they could learn about the missing code snippets strictly from the text and also to highlight the text on which they based their observations. We then developed a labeling scheme to code the observations and analyzed both the codings and other information gained from the frequency and sizes of the code snippets across different document types. Figure 2 provides an illustrative example of steps 1–3.

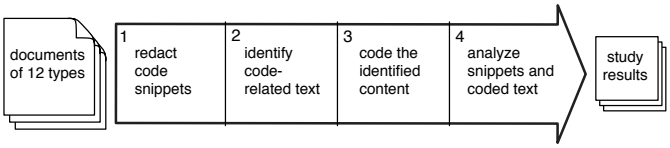


Fig. 1. *Methodology Procedure*

### A. Subjects of Study

Table I specifies the 12 document types that we studied, some example sources from which we sampled the documents, and the unit of granularity that we studied for each document type. We collected 60 document instances, with at least two instances for each type. We randomly selected document instances from 51 distinct sources, all well known, popular sites or high-profile projects. We excluded documents that had no code snippets.

After assembling the corpus, we *redacted* all multi-line code snippets from each document. We redacted the code snippets because we wanted to determine what could be learned about them from only the contextual information in the documents. Redaction was performed using the redaction tools in Adobe Acrobat. Annotators were given documents with each line of a code snippet blacked out or covered entirely with a black rectangle. We did not redact small code snippets that were embedded within paragraphs of text.

TABLE I
SUBJECTS OF STUDY

| Document Type | Example Origin | Unit of Analysis |
|---|---|---|
| Blog Posts | MSDN, Personal blogs | Individual blog posts |
| Benchmarks | OpenMP NAS | One chapter or section |
| Bug Reports | GitHub Issues, Bugzilla | Contents of an issue |
| Code reviews | GitHub Pull Requests, Gerrit | One pull request |
| Course materials | cs.*.edu | One PDF |
| Documentation | readthedocs.org | One documentation page or API |
| E-Books | WikiBooks | Used single book chapter |
| Mailing lists | lkml.org | One conversation thread |
| Papers | ACM Digital Library, IEEE Xplore | Complete paper |
| Presentations | SlideShare | Entire slide deck |
| Public Chat | Gitter, Slack | Single conversation with related snippets |
| Q&A sites | StackOverflow, MSDN | Single question plus answers |

### B. Identifying Code Snippet-related Information

We assigned 37 documents to 35 annotators. The annotators comprised 20 undergraduate, 13 graduate, and 2 professional researchers, all with prior programming experience. We assigned each annotator 3 documents to review. All of the documents were assigned to multiple annotators to review. We received 59 responses from 19 of the annotators (8 undergraduate, 11 graduate, 2 professional). The annotators submitted 49 reviews of 31 distinct documents, constituting of 2-4 documents of each type. 14 of these documents were annotated by multiple people, and in these cases we merged the observations from different annotators.

Each annotator was asked to enumerate as many observations as possible about the redacted code snippets. They also indicated which code snippet they believed was being described when there were multiple redacted code snippets in the same document. They highlighted the text from which they made each observation. Thus, they created an implicit mapping between code snippet, related text, and observation from that text. The annotators made 322 observations in total.

### C. Coding the Identified Relevant Text

Two of the authors identified core code properties that were mentioned throughout the annotators' observations. We defined eight major categories of labels, or codes, for the observed code properties. We further defined subcategories for each of these labels, which we call sublabels, to provide

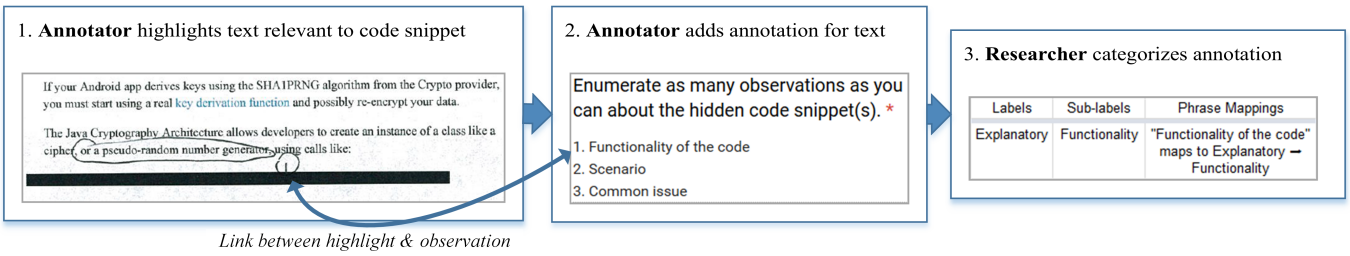


Fig. 2. *Annotation Example*

more detailed categorization of the observations for qualitative analysis. Table II presents the eight labels and their sublabels that we established. Finally, two authors coded each annotator observation with the labels and sublabels based on the occurrence of specific cues embedded in key phrases or key words.

Figure 2 gives an example of a document being analyzed from start to finish. In step 1, an annotator highlights the parts of the document that they think are related to the redacted code snippet. The annotator numbers each highlighted text segment. In step 2, the annotator writes and numbers observations for each highlight. In step 3, we determine a label, sub-label, and the phrases or words in the original highlighted text associated with the observation that cued the labeling.

### *D. Analyzing Snippets and Coded Text*

To compare the number and size of code snippets in each document type, the authors manually counted the number of code snippets in each document and the non-empty lines of code per code snippet. Since document types may vary in the length of a text line, we computed a normalized measure. We first took a sample from each document type and counted the number of characters (including white space and punctuation) to identify the range of number of characters per text line. We then automatically counted the number of characters in the whole document, and divided by an average character count per line for that document type.

We performed two analyses of the coded text and annotators' observations. To determine the kinds of information available in each document type, we computed the frequency of occurrence of the different labels and sublabels. To gain an insight into how the different information is provided for each label, we examined the text highlighted by the annotators to identify the cue words or phrases that most likely triggered the annotator to highlight that information.

## III. Preliminary Results and Discussion

In this section, we provide qualitative and quantitative support for the kinds of information we observed in the various software-related documents. We present preliminary quantitative evidence for the characteristics of the code snippets and related natural language information in each document type considered in our study, as well as the frequency of occurrence of each kind of information we coded. Qualitatively, we present a set of influential word cues found in the different document types we examined, specific for each of the labels and sub-labels we coded.

**Code Snippet Characteristics across Document Types.** For each document type in our study of code snippet characteristics, Table III shows the number of instances in the study (with a total of 60 document instances), the mean number of code snippets in the unit of analysis for that document type, the mean number of lines of code (LOC) in those code snippets, and the mean number of lines of natural language text in the unit of analysis for that document type. Counting the number of code snippets in a document is straightforward except for code reviews.

For code reviews, the code snippets are not embedded in the document; instead, they are being described and discussed and attached to the code review (via hyperlinks). Thus, to count the number of code snippets in a code review, we examined the files that are attached to the review. Each file that is attached is two screens showing a diff. We computed the number of code snippets for a code review by counting each file as 2 code snippets and computed the number of lines of code for each code snippet as the size of the displayed code in the code window, which includes the diff and context lines of code.

Counting the number of lines of code per code snippet is straightforward; however, counting the number of lines of natural language text as a comparable measure across document types is complicated by the fact that some document types have different lengths of text lines. For instance, a research paper could be two columns of text, while a blog post might have a different length of lines than mailing lists. Thus, to compute the number of lines of natural language text, we counted the number of characters of natural language text in each document, and divided that number by 80 as an average length of a line. This provides the information is a more traditional unit of number of lines rather than characters, and normalizes the measure over different document types.

The mean number of code snippets per unit of analysis of each document type varies from 1.2 for public chats to 8.6 for blog posts and research papers. Blog posts, code reviews, research papers, and presentations have the higher number of code snippets ($> 7$), while mailing lists and public chats have only between 1.2 - 2.6 code snippets. Thus, mailing lists and public chats might not prove to be the richest resources for mining code snippets.

The mean length of code snippets varies from 8 lines of code in public chats and documentation to 47 lines of code in mailing lists. In this case of mailing lists, developers often included entire classes or methods to provide context for their

TABLE II
DESCRIPTION OF LABELS AND SUBLABELS WITH EXAMPLE CUES IN TEXT

| Labels | Sub-Labels | Description | Cue Words & Phrases |
|---|---|---|---|
| *Design* | Programming Language | Programming language | Java; C# related names in stacktrace |
| | Framework | Framework used | RxJava; use HorizontalScrollView |
| | Time/Space Complexity | Code complexity | O(nlogn) |
| *Structure* | Data Structure | Data structures or variable types | multidimensional array; cast float as int |
| | Control Flow | Types of control statements used | for loop; if-block |
| | Data Flow | Data flow chains included | data flow chain; TPL data flow usage scenario |
| | Lines of code | Length of code | line 27 follows |
| *Explanatory* | Rationale | Why being implemented in this way | block tridiagonal systems are solved for each direction |
| | Functionality | What is being implemented | perform 3D Fourier transform; this can be done like: |
| | Methodology | How functionality is implemented | with NumCpp; this can be done |
| | Output of code | Results of running code | the end results is |
| | Similarity | Syntactic or semantically similar code blocks | in a similar way; |
| | Modification | Change(s) to existing code | allows you to do the same thing with code like: |
| *Testing* | | Code for testing purposes | depicts such a test method |
| *Origin* | | Origin of code example | adapted from StackOverflow |
| *Clarity* | High | Code is clean and understandable | see how much cleaner it is |
| | Low | Code is unclear or overly complex | it's a mouthful; hard to parse |
| *Efficiency* | Efficient | Better/efficient code example | returns very quickly to its caller |
| | Inefficient | Inefficient code example | has the effect of delaying |
| | Assumptions | Conditions to be met to ensure correctness | only 1-d and 2-d arrays are supported |
| *Erroneous* | Compilation | Code that fails to compile | failed to compile; invalid typecast |
| | Runtime | Contains runtime errors or exceptions thrown | each error is paired with an output |

questions. Many of the document types contain code snippets in the range of 10 to 13 lines of code. This size code snippet may indeed be code examples for single, specific actions, that would be useful to examine further for mining to help developers in different contexts.

As expected, research papers are outliers in size of natural language text in the document with over 400 mean lines of text. However, it is interesting to note that benchmarks, blog posts, and code reviews provide from 65 to 88 mean lines of text, while all the other document types range between 12 to 33 lines of text. This may suggest that research papers, benchmarks, blog posts, and code reviews provide more opportunity for mining descriptive information about the code snippets.

In summary, there are wide variations in the numbers of code snippets, mean size of code snippets and mean lines of natural language text across the different document types. Larger numbers of available code snippets per document provide more opportunity for code snippet mining. More text per document provides more opportunity for descriptive information about the embedded code snippets. The kinds of information that is available in the text could vary, which is the focus of the next section.

**Available Code-related Information in Document Text.** Our human annotators and the coding of their observations were used to gain insight into the kinds of information that the natural language text provides about embedded code snippets in different document types. Specifically, we focused on identifying whether there are certain kinds of information prevalent across many different document types and whether there are kinds of information embedded only in certain kinds of document types. Using the code snippet-related text identified by the human annotators, as well as the observations that they made from that identified text in 31 distinct documents and our labeling of the annotators' observations, we created the heat map in Figure 3. The heatmap illustrates the frequency of occurrence of the different kinds of information indicated by our eight labels for the code snippets in each document type. A darker-colored square indicates a higher frequency of

TABLE III
CODE SNIPPET AND DESCRIPTION AVAILABILITY BY DOCUMENT TYPE

| Document Type | # Docs | Mean # Code Snippets | Mean # LOC Per Snippet | Mean # Lines of Text |
|---|---|---|---|---|
| Benchmarks | 2 | 4.5 | 13.0 | 86.7 |
| Blog Posts | 10 | 8.6 | 13.1 | 88.3 |
| Bug Reports | 6 | 2.7 | 20.1 | 17.2 |
| Code Reviews | 7 | 7.1 | 33.3 | 64.9 |
| Course Materials | 3 | 3 | 12.8 | 12.6 |
| Documentation | 6 | 3.5 | 7.8 | 22.8 |
| E-books | 5 | 2.6 | 21.4 | 37.4 |
| Mailing Lists | 5 | 1.6 | 46.6 | 17.6 |
| Papers | 5 | 8.6 | 10.3 | 439.9 |
| Presentations | 3 | 8.3 | 11.6 | 18.7 |
| Public Chat | 5 | 1.2 | 8.3 | 15.6 |
| Q&A Sites | 3 | 4.7 | 12.6 | 32.8 |
| Total | 60 | | | |

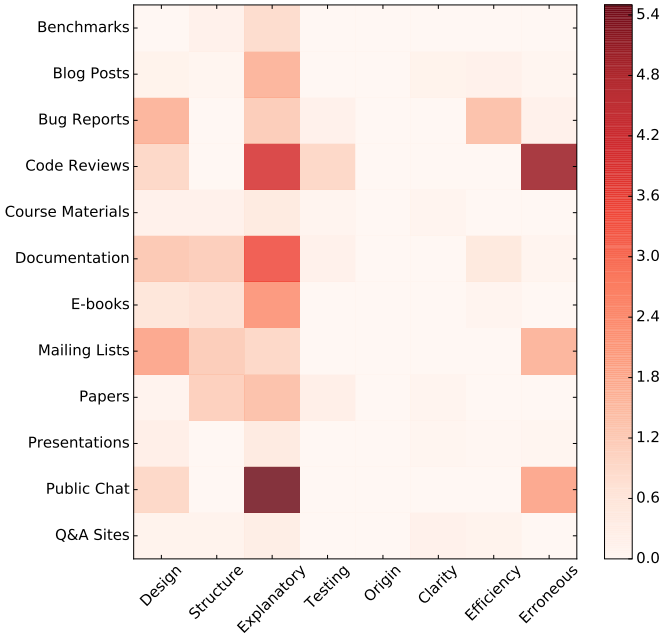


Fig. 3. Heatmap of the frequency of occurrence of different kinds of information for the code snippets in each document type in our study.

occurrence for a particular kind of information in a particular document type, while a lighter-colored square indicates a lower frequency of occurrence. We created the heat map using mean occurrences of each label across the instances of each document type. For example, the four mailing list instances that were coded contained, in aggregate, seven code snippet-related observations that were coded as *Erroneous*, resulting in a mean of $7/4 = 1.75$ occurrences of error-related information in mailing lists.

The heatmap in Figure 3 indicates that Explanatory information is prevalent across most of the document types that we considered, and is the dominant category for several document types, such as blog posts, documentation, e-books, papers, and public chats. This aligns with that fact that the main purpose of these document types is to explain aspects of the implementation in the code snippets. Another kind of information that shows up fairly often across different document types is design information, which includes programming language, framework, and time/space complexity of the code snippet. Based on this exploratory study, information about origin and clarity of the code snippet is rarely available in any of the studied document types. Information about efficiency is found almost exclusively in bug reports, while testing information was found almost exclusively in code reviews.

By examining the heat map from the perspective of an individual document type, bug reports, code reviews, documentation, mailing lists, papers and public chats appear to contain the highest amount of diversity in information about code snippets, relative to other document types.

**Word and Phrase Indicators of Code-related Text.** The third column of Table II depicts examples of words and phrases that appear in the document text highlighted by the annotators. Based on these samples, we note that sometimes individual words are adequate cues for a given kind of information (e.g., a name of a programming language or framework), while sometimes a cue requires a phrase, or sequence of words (e.g., "see how much cleaner it is", "hard to parse"). In fact, for most kinds of information, phrases are needed to cue the kind of information available in the text. These phrases are quite different across information types, which suggests that automating the detection of the kind of information embedded in the natural language text about a code snippet is not trivial. Detection techniques will most likely require natural language processing and machine learning.

## IV. Conclusions and Future Work

In this paper we described an exploratory study in which we investigated the kinds of information available in different software-related documents. We reported our preliminary results, which address research questions about the characteristics of the code snippets embedded in different document types, the kinds of information contained in those code snippets, and the cues that indicate code snippet related information. Of course, a much larger study is needed to definitely answer our research questions. However, this paper provides a methodology for doing so, as well as initial indications of which software-related documents are the most promising candidates for mining descriptions or properties of code snippets at a large scale. Future work includes conducting such a large scale study and using the results to guide the development of mining tools that can extract information about code snippets for a wide variety of software-related documents to support the construction of new models and analyses that enhance software engineering techniques and tools.

## Acknowledgment

We acknowledge the support of the DARPA MUSE program under Air Force Research Lab contract no. FA8750-16-2-0288.

## References

[1] M. Beller, A. Bacchelli, A. Zaidman, and E. Juergens, "Modern code reviews in open-source projects: Which problems do they fix?" in *Proc. 11th Working Conf. Mining Software Repositories*, 2014, pp. 202–211.

[2] R. Tiarks and W. Maalej, "How Does a Typical Tutorial for Mobile Development Look Like?" in *Proc. 11th Working Conf. Mining Software Repositories*, 2014, pp. 272–281.

[3] S. Panichella, J. Aponte, M. D. Penta, A. Marcus, and G. Canfora, "Mining source code descriptions from developer communications," in *Proc. Int'l Conf. Program Comprehension*, June 2012, pp. 63–72.

[4] A. Bacchelli, M. D'Ambros, and M. Lanza, "Extracting source code from e-mails," in *Proc. 18th Int'l Conf. Program Comprehension*, June 2010, pp. 24–33.

[5] C. Vassallo, S. Panichella, M. Di Penta, and G. Canfora, "CODES: Mining source code descriptions from developers discussions," in *Proc. 22nd Int'l Conf. Program Comprehension*, 2014, pp. 106–109.

[6] E. Wong, J. Yang, and L. Tan, "AutoComment: Mining question and answer sites for automatic comment generation," in *Proc. 28th Int'l Conf. Automated Software Engineering*, Nov 2013, pp. 562–567.

[7] C. Treude and M. P. Robillard, "Augmenting API documentation with insights from Stack Overflow," in *Proc. 38th Int'l Conf. Software Engineering*, 2016, pp. 392–403.

[8] G. Petrosyan, M. P. Robillard, and R. De Mori, "Discovering information explaining API types using text classification," in *Proc. 37th Int'l Conf. Software Engineering*, 2015, pp. 869–879.