Crowd Documentation: Exploring the Coverage and the Dynamics of API Discussions on Stack Overflow

Chris Parnin
College of Computing
Georgia Tech
Atlanta, GA USA
chris.parnin@gatech.edu

Christoph Treude, Lars Grammel, and Margaret-Anne Storey
University of Victoria
Victoria, BC Canada
ctreude@uvic.ca,lars.grammel@gmail.com, mstorey@uvic.ca

ABSTRACT

Traditionally, many types of software documentation, such as API documentation, require a process where a few people write for many potential users. The resulting documentation, when it exists, is often of poor quality and lacks sufficient examples and explanations. In this paper, we report on an empirical study to investigate how Question and Answer (Q&A) websites, such as Stack Overflow, facilitate crowd documentation — knowledge that is written by many and read by many. We examine the crowd documentation for three popular APIs: Android, GWT, and the Java programming language. We collect usage data using Google Code Search, and analyze the coverage, quality, and dynamics of the Stack Overflow documentation for these APIs. We find that the crowd is capable of generating a rich source of content with code examples and discussion that is actively viewed and used by many more developers. For example, over 35,000 developers contributed questions and answers about the Android API, covering 87% of the classes. This content has been viewed over 70 million times to date. However, there are shortcomings with crowd documentation, which we identify. In addition to our empirical study, we present future directions and tools that can be leveraged by other researchers and software designers for performing API analytics and mining of crowd documentation.

Categories and Subject Descriptors

D.2.7 [**Software Engineering**]: Distribution, Maintenance, and Enhancement—*Documentation*

General Terms

Documentation, Human Factors

Keywords

Crowd documentation, API documentation, Stack Overflow

1. INTRODUCTION AND MOTIVATION

Software developers need up-to-date and consistent knowledge of their craft and tools. There are many efforts to promote the creation and maintenance of good documentation, however, all too often, documentation is absent or incomplete. When documentation is written, it quickly becomes stale. This stagnation is often the root of mistrust, which can lead to users rarely consulting documentation [11]. Application Programming Interface (API) documentation, in particular, is often spartan [19], leaving developers with insufficient examples or explanations.

Recently, a new culture and philosophy has emerged that is reshaping the face of software documentation. This change has resulted from widely accessible infrastructure and social media technology — wikis, blogs, Q&A sites. Similar to how *open source* development disrupted the process of traditional software development [15], these new forms of contribution and collaboration have the potential to redefine how developers learn, preserve, and communicate knowledge about software development. But unlike *community documentation*, where a person may freely contribute to the documentation of an open source project [3], with *crowd documentation*, the individual contribution through social media does not matter as much as the aggregate result. An individual in a crowd may make a one-time contribution, such as voting on an answer, tagging a question, providing an answer, or asking a question with limited effort and little commitment. For developers who contribute to these new media, there is almost no barrier to entry, no community vetting, and there are no formal processes that would be otherwise associated with traditional or community-based forms of documentation.

To illustrate the potential impact of crowd documentation: while the official documentation for Java provides one code example[1] to illustrate synchronization with `invokeLater()`, but without explanatory text and sparse comments, on the Q&A website Stack Overflow[2], 286 related questions can be found. Not only are the contributions on Stack Overflow more numerous, but the crowd also provides explanatory text, succinct code snippets, multiple perspectives debating the merits of different solutions, votes on popular answers and good questions, and tags for improving searches.

Although we know that social media (and older incarna-

[1] http://java.sun.com/developer/technicalArticles/Threads/swing/Editor.java
[2] http://stackoverflow.com/

tions, such as newgroups) have been actively used by software developers, there has been scant research to investigate the benefits and limitations of social media as a form of software documentation. In this paper, we examine Q&A sites in particular, and we ask whether we can rely on the crowd to generate examples and explanations about APIs. Without a centralized authority, can these many voices produce a comprehensive alternative to traditional documentation? Unlike API documentation, where a collection of resources (such as specifications and code examples) are comprehensively organized and explicitly linked with API elements, crowd documentation is created in a tacit manner, without any comprehensive organization or explicit links to API elements. Although we can offhandedly infer the value of crowd documentation by the large number of contributors and contributions present, we have no comprehensive way for understanding the content of contributions or for evaluating the value and quality of those contributions.

To explore the feasibility of crowd documentation for software development, we examine the coverage and dynamics of the discussions about APIs on Stack Overflow. To analyze coverage, we build a traceability model between API elements and the questions and answers on Stack Overflow, and then measure the percentage of classes that have traceability links into the examples and explanations generated by the crowd. A high coverage would suggest that it is feasible for crowd documentation to generate a comprehensive source of knowledge about an API. To analyze the dynamics, we examine the nature of the roles, contributions, and curating actions of the crowd. A high number of contributors would confirm the "crowd-like" nature of crowd documentation and a high-level of knowledge curation would suggest improved quality of crowd documentation. Further, understanding the dynamics of crowd documentation can give us insights into the chemistry of successful and unsuccessful communities on Q&A sites.

To perform our empirical study, we collected evidence from over 7 million question and answers on Stack Overflow accumulated over a 4 year span, and studied in detail: 6,323 questions and 10,638 answers related to the GWT API, 119,894 questions and 178,084 answers related to the Android API, and 181,560 questions and 445,934 answers related to the Java API. We also collected usage data for 2,432 classes of the Java API and for 1,038 classes of the Android API through Google Code Search.

We find that crowd documentation can generate many examples and explanations of API elements. Although the crowd can achieve a high coverage of API elements, the *speed* of achieving that coverage was linear over time. Furthermore, sometimes the crowd may need to be steered towards certain topics. For example, the crowd was mute on certain topics, such as accessibility (*e.g.* accommodate disabled users) or digital rights management (DRM) capabilities in Android. We also found that the discussion dynamics follow a pattern where the "crowd" asks questions and a smaller pool of "experts" answer them. However, we observed that the least active API, GWT, lacked this dynamic, suggesting that API stakeholders may want to play a stronger role in seeding communities.

Finally, for the purpose of galvanizing the research community, we extend our focus beyond passively observing crowd documentation and outline future steps and directions for actively harnessing crowd documentation in applications of *crowd documentation mining and analytics*. For example, we illustrate a simple approach for using the crowd-curated discussions on Stack Overflow to automatically generate API documentation for a class. We also provide a treemap visualization tool for exploring the discussion and usage of API elements, allowing API stakeholders to assess the health of their API through the voice of the crowd.

After reviewing related work in Section 2, we make the following contributions:

- a conceptualization of crowd documentation (Section 3),
- a foundational study on the coverage and dynamics of crowd documentation (Sections 4 and 5), and
- future directions and approaches for crowd documentation mining and analytics (Section 6).

2. BACKGROUND AND RELATED WORK

Work related to our research can be divided into work on API documentation and recent work on using social media channels for documenting software.

2.1 API Documentation

The knowledge needed by software developers is diverse. It is captured using different forms of documentation, written by different creators, and communicated to a variety of audiences [22]. Several researchers have strongly argued that documentation should be written from multiple views and perspectives [2] which span requirements, behaviors, tasks, and should include patterns and examples [8].

Many developers regard documentation as a necessary evil, written only as an afterthought to adhere to bureaucratic regulations [17]. Therefore, documentation ends up being incomplete and inaccurate. In a survey of software professionals, Forward and Lethbridge [5] found that content, up-to-dateness, availability, and the use of examples are the most important document attributes. In a related study, Lethbridge *et al.* [12] explored how software engineers use and maintain documentation. They found that most software engineers do not update documentation in a timely manner, with the exception of highly-structured, easily-maintained forms of documentation, such as test cases and in-line comments.

The nature of the documentation process differs based on the context. In *closed documentation*, documents rarely leave the boundaries of a closed source system: documents are created by a few and read by a few. In contrast with closed documentation, *API documentation* is written by a few, but read by many. For example, with the Java programming language, developers can refer to JavaDoc documentation of classes and methods to understand how to use the API. The process of documenting APIs and frameworks requires different kinds of documents [1], including example applications, recipes and cookbooks, patterns, motifs, contracts, overviews, and reference manuals.

Effectively documenting and using APIs is not trivial. Robillard [19] observed insufficient or inadequate examples as obstacles for developers trying to learn an API. He also identified several other task-, format-, and design-related obstacles. Although useful documentation can come from JavaDoc, the resulting documentation is often narrow in focus and sometimes overly verbose.

These shortcomings have called the software engineering research community to action. But rather than improving

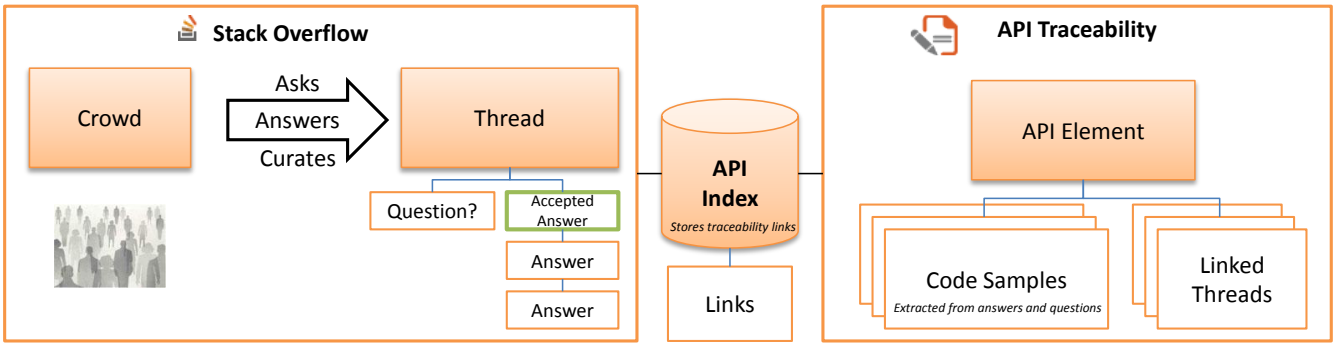

Figure 1: Our model for crowd documentation of APIs.

the process of creating API or framework documentation, the solution to bad documentation has been to instead build better knowledge discovery tools [4]. Research tools, such as Hipikat [23] or Strathcona [6], provide developers with recommendations in the form of relevant artifacts or code examples. Other tools, such as jungleloids [14], ease API use by answering how to properly sequence a set of calls in order to reach a certain property or object.

2.2 Socially-Mediated Documentation

The possibilities and limitations of crowd documentation in software development are currently unrealized. Guided by research that found web search to be the leading way to learn about new APIs [20], in our previous work [18], we analyzed the Google search results for one particular API – jQuery – and discovered that, besides the official sources of documentation, many socially-mediated sources of documentation appeared. For example, we found at least one blog post on the first page of the search results for 88% of the methods in jQuery; and for 84% of the methods, we found at least one question on Stack Overflow on the first page of the search results.

Pagano and Maalej examined the blogging behaviors of developers who commit to open source projects: Eclipse, GNOME, PostgreSQL, and Python [16]. They used a blog aggregator site to find bloggers associated with the development projects, and then matched the blogger's identity to source code committers in the project. In the study, only 1.8% of blog posts contained source code. Pagano and Maalej concluded that rather than documentating code, most open-source developers used blogs for communicating and coordinating functional requirements and domain concepts about the project (as determined by an automated LDA topic modeling algorithm) with the community and other developers.

In their study on the design elements behind the Q&A website Stack Overflow, Mamykina *et al.* [13] found that its success is not just due to technical design, but also due to curating activities (such as voting) and incentives (such as reputation scores). They also found that Stack Overflow users get very fast responses: questions are answered in a median 11 minutes. In our own preliminary work [21], we categorized the kinds of questions asked on Stack Overflow. We found that the website is particularly effective at code reviews and conceptual questions, and that roughly 85% of the questions on Stack Overflow are answered. In a similar but much earlier study, Johnson and Erdem [9] found that questions on Usenet newsgroups were either goal, problem, or system oriented. Jiau and Yang searched the Stack Overflow question titles for 103 UI widget classes and found that some classes were discussed much more frequently than others [7].

Previous work on Stack Overflow has mostly ignored the *content* of social media sites, instead focusing on the *mechanics* of discussion. In this work, we want to look beyond the mechanics of social media, and begin to understand the content and value these social activities bear. To do so, we look at the traceability links between API elements, questions, and answers on Stack Overflow, as well as the roles and contributions of the crowd toward generating knowledge about those API elements. From this understanding, we gain better insight into how social media can be used as a platform for socially-mediated documentation of APIs, *i.e.*, the crowd documentation of APIs.

3. CROWD DOCUMENTATION

In this section, we describe a model that formally defines the elements of crowd documentation, including the roles and contributions of the crowd and their relationship to API elements (see Figure 1 for an overview). We feel that our model provides a variety of benefits. For example, other types of social media, such as web forums or blog sites, can be compared and contrasted to our model. A consistent terminology can be established for future researchers exploring this topic. Finally, the model is instructive for readers unfamiliar with the mechanics of Stack Overflow.

3.1 A Model of Crowd Documentation

Our model consists of threads, contributors along with their roles and actions, and links between threads and API elements.

3.1.1 Threads

Crowd documentation is a collection of web resources, where a large group of contributors, the *crowd*, curate and contribute to the collection. A *crowd document* is a crowd-curated web resource that is distinct from other web resources, such as source code repositories, where curation is less prevalent. On Stack Overflow, the primary type of a crowd document is a *thread*, a question with a collection of answers (see Figure 2 for an example). A *question* is an entry composed of a title, question body, and a maximum of five tags. An *answer* is an entry composed of a body of

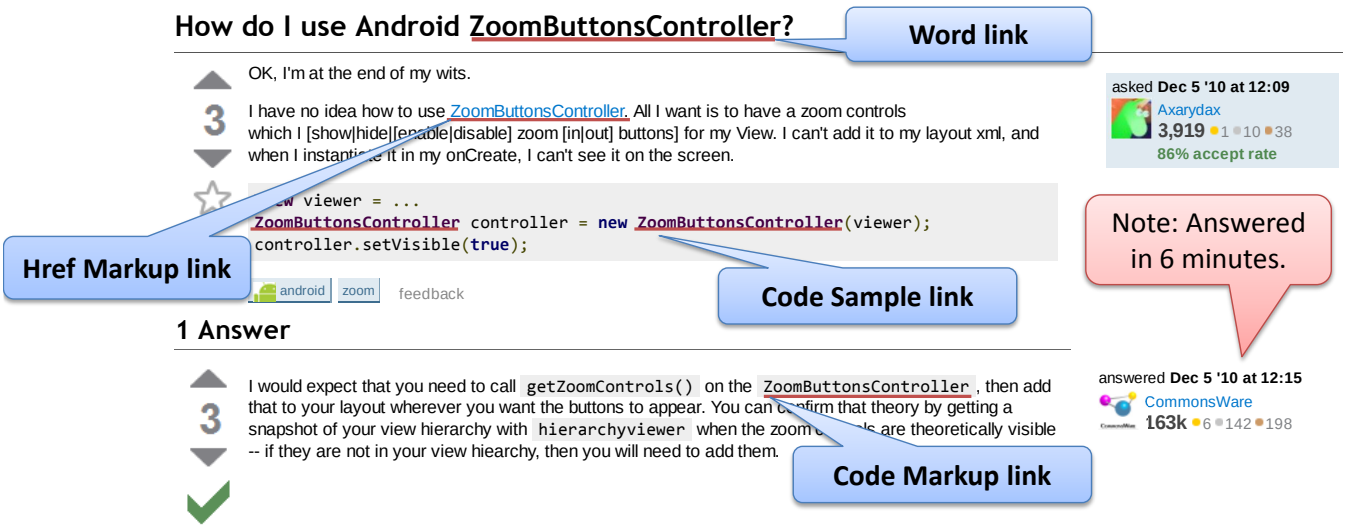

Figure 2: A Stack Overflow thread annotated with different class reference link types.

text, providing a definite resolution to the question that is intended for clarification or further discussion.

3.1.2 *Contributors and Roles*

A *contributor* is a member of the crowd, fulfilling at least one contributor role. A *contributor role* is a capability for performing actions on crowd documents, which may be limited based on the member's credentials. For Stack Overflow, the following contributor roles are available: asker, advisor, and curator. A contributor can take on multiple roles. An *asker* provides questions for the crowd and is the only contribution role capable of designating an answer as "accepted". An *advisor* provides answers to a question. A *curator* can perform several actions, such as voting on a question or answer, or marking a thread as a "favorite".

3.1.3 *Contributor Actions*

Contributor actions are commands that can be performed on a thread or its elements, allowing contributors to curate crowd documents. The following actions are available to contributors on Stack Overflow:

ask An asker posts a new question.

answer An advisor provides an answer to a question.

accept An asker designates an answer as having resolved the question.

favorite A curator saves a thread in their personal list of threads.

vote A curator upvotes or downvotes the score of an answer or question. Scores contribute to a contributor's reputation.

view A public visitor views a thread.

bounty A curator pays a bounty (an amount subtracted from his or her reputation) to the advisor providing a good answer.

3.1.4 *Links*

An *API element* is a named entity belonging to an API, such as a class, interface, or method. In order to identify the crowd documents related to an API element, we maintain an *API Index*, a dataset containing *traceability links*, a connection between an API element and a crowd document. Traceability links associate a set of code samples and threads contributed by the crowd to an API element.

In order to collect more information about where and how traceability links occur, traceability links record the site (such as in an answer or question) and span (character positions) of the match. The site of a traceability link is used to determine its type:

word link A match occurring in the text of a thread.

code markup link A match fully enclosed by `<code></code>`.

href markup link A match fully enclosed by `<a></a>`.

code sample link A match occurring in the text of a `<code></code>`.

In Figure 2, several types of traceability links to the `ZoomButtonsController` API element from Android are shown.

4. RESEARCH METHODOLOGY

This section outlines our research questions as well as our data collection and analysis methods.

4.1 Research Questions

To determine the value of crowd documentation, it is crucial to understand whether we can rely on the crowd to provide questions and answers for an entire API on Stack Overflow in a reasonable amount of time. To answer this question, several aspects have to be considered: the extent to which the crowd discusses different API elements, which areas of an API are covered (and which ones are not), and the speed at which this coverage is achieved. When developers contribute to Stack Overflow, how often will API elements be included in those discussions? Are there some API

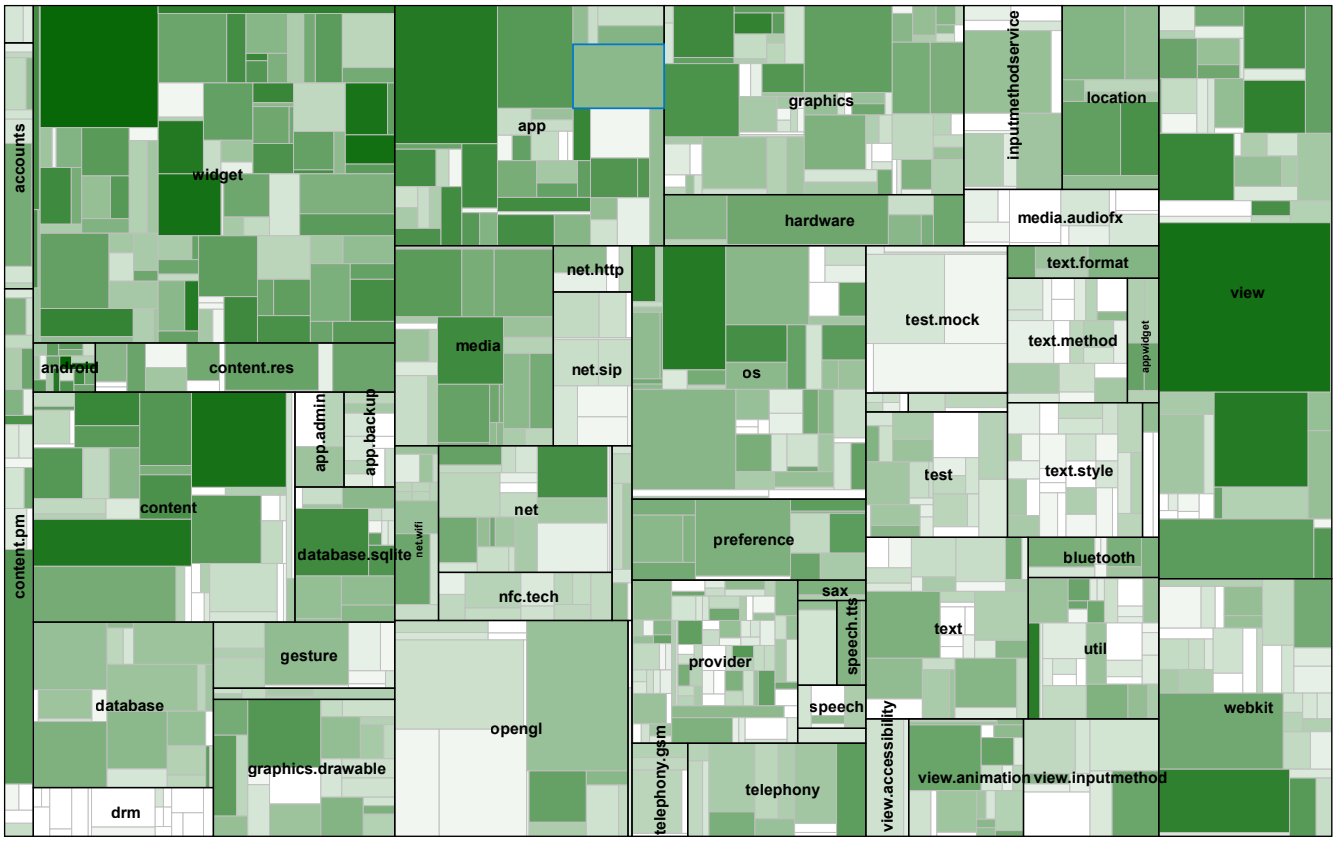

Figure 3: Crowd documentation of Android classes. Packages and their classes are shown as a treemap, and the number of methods is used to calculate the size for each class. A color scale is used to indicate the number of threads: white classes have no threads whereas dark classes have many threads. A log scale is used for mapping the number of threads onto color.

elements that are ignored, and are there others that are frequently discussed? One of the purported advantages of any crowd-sourced effort is the immense speed and scalability that the crowd can supposedly achieve. For organizations wanting to forgo or facilitate the production of API documentation, to what extent can API stakeholders rely on crowd documentation? Will it be quick enough?

Therefore, with our first main research question, we ask:

RQ1. Can we rely on the crowd to discuss an entire API on Stack Overflow?

RQ1.1 Are APIs widely covered?

RQ1.2 What is discussed and what is not discussed? Are those API elements infrequently discussed also infrequently used in practice?

RQ1.3 How fast is the crowd at covering an entire API?

Unlike traditional documentation, crowd documentation has to rely on a functioning crowd to provide questions, answers, votes, and comments. The availability of such a crowd and its good dynamics are essential for crowd documentation. To examine the dynamics of a successful API community on Stack Overflow, we need to understand who the contributors are and what contributions they make. The crowd has a variety of contributer actions available to them for designating the quality of answers and questions on Stack Overflow. Some are implicit (e.g., viewing a thread), some are personal (e.g., marking a thread as "favorite"), and some are paid (e.g., offering a bounty). How are these various curator operators applied by the crowd, and what effect do they have on the resulting coverage of crowd documentation?

Several research efforts have suggested that documentation can be improved by finding code examples from web resources. However, the specific nature of the question and answer format on Stack Overflow may change that dynamic. For example, when compared with code samples found in code repositories (such as GitHub) or in web resources (such as blog posts), code samples in Stack Overflow questions may be more likely to contain problems. Is there still a chance for success if we rely on the dynamics of a crowd?

Our second main research question focuses on these different aspects of crowd dynamics:

RQ2. What are the dynamics of a successful API community on Stack Overflow?

RQ2.1 Who contributes?

RQ2.2 How does the crowd contribute?

RQ2.3 How many code samples does the crowd provide?

API	Tagged	Answered	Linked	Accepted	Voted	Viewed >500	Favorited	Bountied
Android	119,893	102,160	70,123	62,497	47,896	26,285	25,782	970
		85%	58%	52%	40%	22%	21%	1%
Java	181,559	171,886,	107,184	115,169	106,462	45,159	44,280	1,320
		95%	59%	63%	58%	25%	24%	1%
GWT	6,322	5,787	3,277	3,673	3,185	1,779	1,584	49
		92%	52%	58%	50%	28%	25%	1%

Table 1: Breakdown of Q&A threads on Stack Overflow that are tagged with "android", "java", or "gwt".

4.2 Data Collection

To retrieve data from Stack Overflow, we downloaded the Creative Commons Data Dump that the Stack Overflow team makes available on their blog[3]. While we piloted our methodology with several versions of this dataset, all the results we present here are based on the December 2011 version unless otherwise noted. We imported the data into a relational database. For each question and each answer on Stack Overflow, we obtained information such as the title and body, the tags assigned to the question, the creation time stamp as well as the time stamp of the last edit, and the number of times somebody had viewed the contribution. We also retrieved the score for the contribution as the aggregation of up-votes and down-votes, the number of times somebody had marked a contribution as "favorite", whether an answer had been accepted, who wrote and edited a contribution, and how many answers and comments there were per question. For each user on Stack Overflow, we obtained their display name, registration date, age and location, as well as their reputation score (that is calculated based on their activity on the site).

We examined the crowd documentation of three popular APIs: the Android API[4], the GWT API[5], and the API of the Java programming language[6]. We selected these APIs because they have different characteristics, and yet, are easy to compare. While all are written in Java, the Java API is large and well-established, whereas the Android API is young and only applies to the limited domain of mobile development. Finally, while the GWT and Android APIs are comparable in size, they have different levels of activity on Stack Overflow. For all APIs, we downloaded a list of all packages and classes: GWT had 845 classes, Android 1,038 classes, and Java 2,432 classes.

In addition, we collected usage data for the APIs using Google Code Search[7]. For each class in our data set, we queried Google Code Search for import statements containing that particular class (e.g., to get usage data for the Android class TextView, we queried Google Code Search for "import android.widget.TextView"). Since most modern code editors support automatic import organization, this offers a good approximation of actual usage data. Generic imports, such as "import android.widget.*", are rarely used[8]. Unfortunately, we could not collect usage data for GWT because of the recent closure of the Google Code Search API.

4.3 Data Analysis

To build our explicit model of crowd documentation for API classes, we mined the Stack Overflow data for traceability links to the classes. To guide our mining efforts, we only considered threads tagged "android", "java", or "gwt".

Traceability links are found using case-sensitive, word boundary matches. Additionally, several semantic rules are applied to ensure correct matches. In the case of name collisions, we use the fully qualified name. For example, for the Java `Date` class, we use fully qualified names to distinguish `java.util.Date` and `java.sql.Date`. There are also a few cases where word boundaries will not correctly distinguish classes. For example, in Android, using naive word boundaries, `AbsListView.SelectionBoundsAdjuster` would also match `AbsListView`. To avoid this problem, we use look ahead matching in certain cases. Finally, to avoid collisions with common English words, we exclude word links for single-word API elements (as defined by camel case), such as the `Intent` class in Android.

To identify the traceability link type, we parse the question or answer body using jSoup[9]. To extract a code sample, we collect all the code blocks in the match site that are not associated with code markup links from the question or answer body. For each class, we then generate a list of threads and a list of extracted code samples associated with the class.

For researchers wanting to perform similar studies, we have made our code and the generated crowd documentation of API classes (serialized as json files) available for download at `https://github.com/chrisparnin/anacrowd`.

4.4 Characterizating the Collected Data

To give insight into our collected data, we provide a brief characterization.

4.4.1 Threads

For all APIs, over half of the threads discussed at least one API element, and most threads were curated in some manner. For the threads we considered, 91% of threads have at least one answer, with 64% of them accepted by the asker. Many threads are also voted, viewed frequently, and favorited by users. Table 1 shows detailed breakdowns of the collected data.

4.4.2 Links

Most traceability links are code sample links (73%), followed by word links (15%), code markup links (8%), and href markup links (4%).

[3]http://blog.stackoverflow.com/category/cc-wiki-dump/
[4]http://developer.android.com/reference/packages.html
[5]http://google-web-toolkit.googlecode.com/svn/javadoc/1.6/index.html
[6]http://download.oracle.com/javase/6/docs/api/
[7]http://www.google.com/codesearch
[8]For example, Google Code Search finds 52,790 import statements with the fully qualified name of TextView compared to 2,023 search results for the generic import of the android.widget package.
[9]http://jsoup.org/

Many traceability links occur in questions. 53% of threads are linked due to traceability links in the question. In contrast, 38% of threads are linked due to traceability links in an answer, and 9% due to traceability links in both answers and the question. When inspecting threads, we observed two types of patterns that can explain this mutual exclusion. In the first pattern, an asker would present a problem and associated code demonstrating the problem. Advisors would then provide an explanation that would resolve the problem, but not necessarily with code. In the second pattern, an asker would ask for advice on the feasibility or the best way to achieve a task with an API. Advisors would then provide different code samples that would accomplish the asker's goal. Finally, 21% of threads have traceability links in the accepted answer.

4.4.3 Views

According to our data, the 307,774 threads from Android, Java, and GWT have been viewed a total of 200 million times!

5. FINDINGS

5.1 Reliability of the Crowd

In the following, we present the findings to our first research question, **RQ1. Can we rely on the crowd to discuss an entire API on Stack Overflow?**

5.1.1 Extent of Coverage by the Crowd

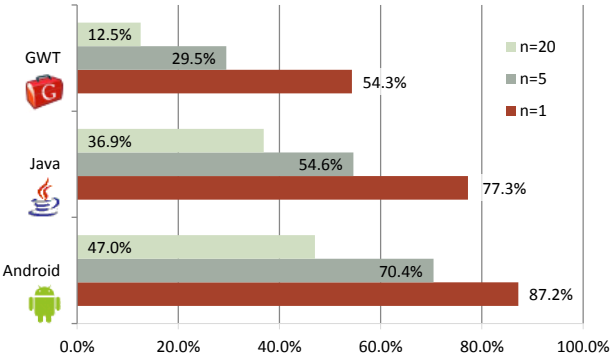

Figure 4: Coverage of API elements in crowd documentation of three APIs for three different levels of saturation (# threads per API element).

RQ1.1 Are APIs widely covered?

To answer our first research question, we examined the crowd documentation's coverage of API elements. *Coverage* is the percentage of classes that have at least `n` threads discussing the class, where `n` is called the saturation level. To understand how coverage varied across different APIs, we examined the coverage of the three selected APIs on Stack Overflow: GWT, Android, and Java. Figure 4 shows the results.

For 87% of all classes of the Android API as well as for 77% of all classes of the Java API, we found at least one thread on Stack Overflow. Although the number of Java classes is more than twice the number of Android classes, both reached a comparable level of coverage. In contrast, even though GWT is slightly smaller than Android, it has a lower level of coverage at each saturation level. Low activity on Stack Overflow can impact the comprehensiveness of crowd documentation—only 3,277 threads discussed GWT API classes in comparison to 70,123 threads discussing Android API classes. Finally, coverage with higher levels of saturation (*i.e.*, at least 20 threads per class) is lower, but still comes in at 47% for Android, 37% for Java, and 12% for GWT.

5.1.2 Areas Covered by the Crowd

RQ1.2 What is discussed and what is not discussed? Are those API elements infrequently discussed also infrequently used in practice?

We examined our second research question in two steps: In a first step, we analyzed the correlation between how often certain classes are used and how often they are mentioned on Stack Overflow to understand whether low coverage for particular elements can be explained by low usage in general. In a second step, we conducted a manual inspection of the most and least covered packages of each API to see what kind of classes are discussed a lot, and which ones are not discussed at all.

For both Android and Java, we found a strong correlation between usage data (from Google Code Search) and coverage data (from Stack Overflow): A Spearman's rank correlation coefficient of 0.797 for Android, and a Spearman's rank correlation coefficient of 0.772 for Java. Classes that are used by many developers are likely to have a high discussion volume on Stack Overflow, and classes that are only used infrequently in practice are not likely to be well documented by the crowd either. We were not able to obtain usage data for GWT because of the recent closure of Google Code Search.

To do a more detailed analysis of areas that are thoroughly discussed on Stack Overflow, and areas that are largely ignored by the crowd, we developed a visualization that shows the number of threads for each class of an API. Figure 3 shows the result for the Android API. Packages and their classes are shown as a treemap, and the number of methods determines the size for each class. A logarithmic color scale is used to indicate the number of threads: white classes have no threads whereas dark classes have many threads. Popular packages such as android.widget and android.view are well covered, whereas areas such as the digital rights management (android.drm) and accessibility (android.accessibilityservice) are largely ignored by the crowd. An interactive version of the visualization is available online for both Android[10] and Java[11]. For Java, popular packages such as java.util and java.swing are covered well, whereas areas such as java.security are largely ignored.

5.1.3 Speed of the Crowd

RQ1.3 How fast is the crowd at covering an entire API?

To answer this question, we examined the speed and coverage over time of crowd documentation. To measure speed, we looked at how many API elements were discussed at every date between July 31, 2008 and December 1, 2011 for

[10] `http://latest.crowd-documentation.appspot.com?api=android/`

[11] `http://latest-java.crowd-documentation.appspot.com/?api=java`

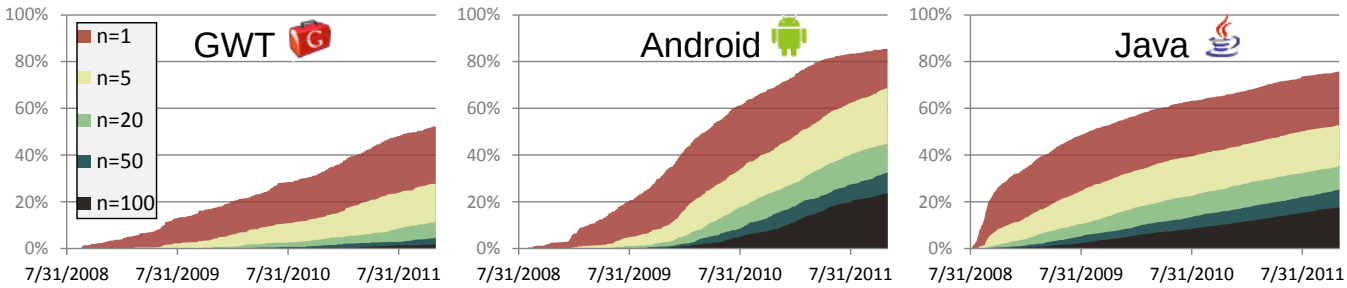

Figure 5: API coverage over time at various saturation levels for GWT, Android, and Java.

Android, Java, and GWT. To measure coverage, we calculated coverage of classes with various saturation levels (n = 1, 5, 20, 50, 100 threads) over time.

The results for the three APIs can be seen in Figure 5. For all three APIs, the rate at which new classes are covered by the crowd with various saturation levels follows a linear pattern, with the exception of Java in the first year. The crowd was very fast at discussing new Java classes in the first year of Stack Overflow (about half of all classes in the Java API were mentioned on Stack Overflow by the end of July 2009), but the speed of the crowd decreased after that.

5.1.4 *Summary*

API designers cannot completely rely on the crowd to provide questions and answers for an entire API. While there is at least one Q&A thread on Stack Overflow for about 80% of the classes of the Java and Android APIs, some areas (such as accessibility) are ignored by the crowd. Also, the crowd takes time to discuss all classes of an API at a linear rate, focusing on API elements that are frequently used.

5.2 Dynamics of the Crowd

In the following, we present the findings to our second research question, **RQ2. What are the dynamics of a successful API community on Stack Overflow?**

5.2.1 *Crowd Contributors*

RQ2.1 Who contributes?

To answer this question, we examined the size and composition of the crowd and its contributions in GWT, Android, and Java. Our goal was to identify any distinguishing characteristics that might reveal an interesting dynamic in how the crowd made its contributions. The results for the crowd size are as follows: 1,899 askers and 2,386 advisors in GWT; 25,065 askers and 21,063 advisors in Android; 44,867 askers and 45,992 advisors in Java. To determine composition, we looked at each user and each thread and counted the number of times a user acted in an asker or advisor role.

To assist in characterizing the composition, we used percentile ranks to bin segments of users. Percentile ranks are determined by sorting users by their frequency of contribution. Figure 6 shows the distribution of askers and advisors for all APIs. For all APIs, the most active 25% askers make up for more than 60% of the questions. For advisors, the distribution is more skewed; for Java and Android, the top 25% advisors contribute around 85% of the answers. The numbers are even more extreme when we only look at the top 5% advisors; they contribute 64% of the answers for Java, and 59% of the answers for Android. Consistently across all APIs, the distribution for askers is not as skewed as for advisors, suggesting that questions come from a wide array of people, whereas answers are largely influenced by a small group of "power users".

Filter	GWT	Android	Java
Coverage	**459 (54%)**	**905 (87%)**	**1879 (77%)**
accepted	427 (51%)	861 (83%)	1773 (73%)
answered	458 (54%)	898 (87%)	1865 (77%)
favorited	361 (43%)	817 (79%)	1560 (64%)
voted >= 3	306 (36%)	736 (71%)	1531 (63%)
viewed > 500	366 (43%)	807 (78%)	1586 (65%)
bountied	80 (9%)	434 (42%)	750 (31%)
at least 2 filters	446 (53%)	884 (85%)	1827 (75%)
at least 3 filters	387 (46%)	834 (80%)	1680 (69%)
at least 5 filters	237 (28%)	641 (62%)	1182 (49%)

Table 2: Effects of applying different filters on the API coverage.

We did observe that the distribution of the GWT advisors was not as strongly skewed as the distribution of Java and Android advisors, providing a possible explanation for the relatively low coverage of GWT.

5.2.2 *Crowd Curation*

RQ2.2 How does the crowd contribute?

To answer this research question, we examine the effect of using various curation filters (based on available curation actions) on the crowd documentation model. Unlike traditional documentation, crowd documentation can be filtered using various quality attributes. Each thread on Stack Overflow has a number of quantitative properties, such as the number of views, the number of votes (and the score based on these votes), whether it has been answered and whether an answer has been accepted and/or bountied, and how many times it has been favorited. While some of these curation filters are explicit (such as votes), others are implicit (such as views). Stack Overflow uses these filters to provide sorting of Q&A threads, e.g., threads on Stack Overflow can be sorted by votes and level of activity. To ensure a certain level of quality (e.g., only consider threads with an accepted answer), the crowd documentation can be filtered to only include threads that meet a certain threshold.

To measure the effect of each curation filter, we measured the change in coverage for each API if only threads meeting a certain filter are considered. For example, for the filter "viewed > 500", we excluded threads that have been viewed less than 500 times. Table 2 shows the results. Apart from

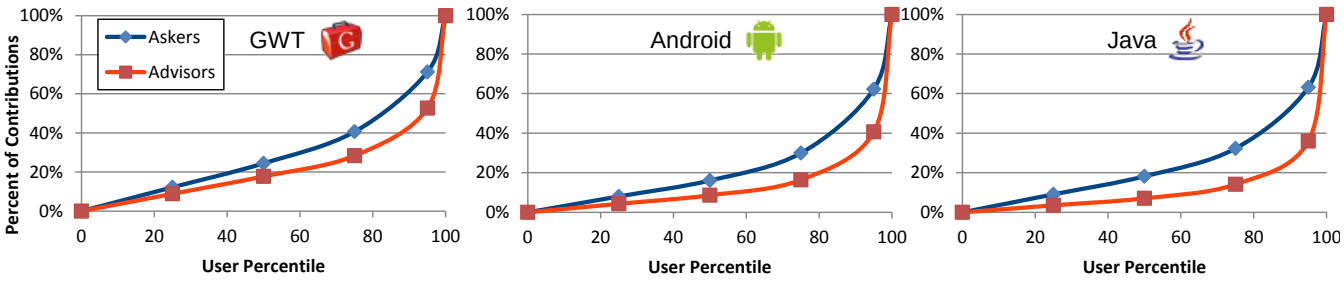

Figure 6: The long tail distributions of askers and advisors for GWT, Android, and Java.

Metric	GWT	Android	Java
classes with code in question	34%	74%	61%
classes with code in answer	33%	66%	58%
classes with code in accepted ans.	26%	56%	49%
classes with code total	43%	78%	68%
samples per class in question	3	125	59
samples per class in answer	2	53	52
samples per class in accepted ans.	1	23	19
samples per class total	5	178	111

Table 3: Distribution of code samples across APIs and location.

the filter that requires a bountied answer, none of the individual filters decrease the coverage by more than 20%.

5.2.3 *Code Samples*

RQ2.3 How many code samples does the crowd provide?

To answer this question, we examined the number of code samples that we linked to API classes. Also, we categorized code samples based on the location they appeared in (i.e., questions, answers, and accepted answers). As code samples in questions can contain problems (e.g., copied error messages [21]), code samples in answers are likely more useful. To ensure we were not matching very small code snippets, we only considered code samples where a code block contained more than one line and the total code sample contained at least ten lines in size.

Table 3 shows the distribution of code samples across locations for each of the APIs in our data. For Java and Android, at least 49% of the classes have at least one code sample in an accepted answer, averaging 23 and 19 code samples per class, respectively. The numbers for code samples in answers (including non-accepted ones) and questions are higher, with 78% of the Android API classes having at least one code sample on Stack Overflow.

5.2.4 *Summary*

The composition and dynamics of the crowd suggests that a "crowd" of developers asks questions and a small pool of "experts" answer them. However, this dynamic may break down if experts are making insufficient contributions. Crowd documentation can be filtered in a number of ways to ensure quality. Most filters only slightly decrease the coverage, and still yield crowd documentation that covers more than 60% of the Android and Java APIs. Code samples can be mined from crowd documentation. Stack Overflow has at least one code sample in an accepted answer for about half of the classes of popular APIs.

6. DISCUSSION AND FUTURE DIRECTIONS

This section discusses our findings and their implications.

6.1 Steering the Crowd

The crowd is too slow to ever replace official API documentation. For example, within the first year of Stack Overflow's operation, the coverage of Android API classes only reached 30%. After time, the situation greatly improved, but still left pockets of uncovered areas. However, there are several possible ways to help steer the crowd, for example, by injecting incentives into the crowd. Because participation can be driven by reward [13], API designers may be able to use these incentive channels to directly recruit the crowd to help them out. They can reward bounties (reputation points) for providing questions and answers to specific API packages or tasks. Automated badges (achievements that can be earned by Stack Overflow users) can automatically be rewarded to those that first ask or answer a question about a certain API element. Reputation can be a powerful incentive: Highly reputable Stack Overflow users have been contacted by software company recruiters based on their reputation, or received offers for consulting business. It is no wonder that the highest contributer to the Android questions and answers also runs an Android consulting business.

Finally, API stakeholders trying to establish an API may find it worthwhile to detect and invest expertise when an insufficient number of expert contributions are being made.

6.2 Mining for Code Samples

There have been several approaches for improving documentation by automatically locating code examples. For example, JavaDoc generally does not include code examples; as a result, researchers have tried to automatically locate and extract examples of how to use a method from code on the web [10]. A weakness of this approach is that the extracted code snippets have no context explaining what they are trying to achieve. By mining code snippets created by the crowd, this weakness can be largely offset from the explanatory text and discussion accompanying the code snippet.

We have demonstrated how API classes can be automatically associated with code samples and highly rated threads. For example, code samples in accepted answers could be found for over half of the classes. However, many code examples also exist within the body of a question. In many

cases, these types of code samples may not be "ready-made" for automated extraction, but are only understood by a human reading the discourse and guidance provided by the advisors in a thread. There may be numerous opportunities for more sophisticated mining techniques that can merge and auto-correct problematic code in the question. Further, recommendation systems can use problems and resolutions reported in threads as a knowledge base for programmers experiencing the same problem in an IDE.

Another unexpected challenge may be that for certain API elements, we have actually *too many code examples*. For example, `java.util.ArrayList` is a very popular and widely used class featured in 8,315 code examples. How do we find the best code examples and questions about `ArrayList`?

Finally, instead of just *finding* code examples, can we automatically *generate* the documentation itself? For example, a format similar to JavaDoc could be generated, including popular questions about a class, recommendations to external resources such as blogs, and popular code snippets. See an example created automatically by our prototype tool:
`http://se.ninlabs.com/exp/crowd/examples/MessageDigest.html`

6.3 Crowd Documentation Analytics

For better understanding of crowd documentation, we developed an interactive treemap visualization that can be freely configured to show the number of threads, usage, or number of methods for the classes of an API using size and color mappings. In addition, the visualization can be filtered by user name to highlight the contributions of a particular user. This allows researchers and API designers to understand their community and even allows users to visualize their contributions amidst the crowd. For example, potential areas of difficulty, as identified by a disproportionate ratio of discussion in questions and usage in practice, or large patches of undiscussed sections of APIs could be detected and explored. The interactive visualization is available online for Android and Java at:
`http://latest-print.crowd-documentation.appspot.com/?api=android`
`http://latest-print.crowd-documentation.appspot.com/?api=java`

In Figure 7, we display a zoomed selection of Android, with the usage and discussion ratio color filter. Note that classes in some packages, such as `database.sqlite`, may be problematic for programmers, as indicated by dark green classes. Inspecting the related threads suggests this may be due to many exceptions that developers experience.

7. LIMITATIONS

As with any chosen research methodology, there are limitations with our choice of research methods.

Our traceability analysis may underestimate the number of links. For example, we would exclude a class name that was not matching the exact case (`jsonparser`) or with spaces placed in between words (`Zoom Buttons Controller`).

In this paper, we focused on API elements that were classes and our results may not generalize to other elements such as methods or other aspects of APIs such as installation or deployment. Even though the crowd curates discussions on Stack Overflow, we do not quantify the relative strength of a particular type of curation. For example, is it more important that a thread is frequently viewed or highly voted? However, we argue that mining and analyzing the discussions of classes on Stack Overflow represents an important and significant step.

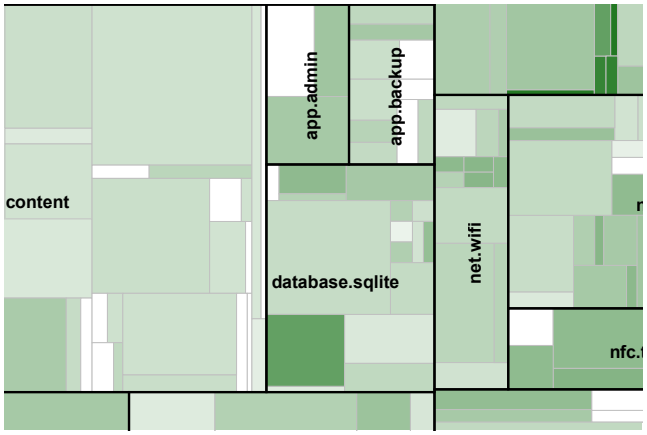

Figure 7: Zoomed selection of treemap for Android, with usage and discussion ratio color filter. Areas in dark green may indicate problematic classes.

We also only examined our research questions from the perspective of three APIs. Although they provide different perspectives (a large established API, a new and specialized API, and a less active API), we do not know how well our results extend to other APIs that are more difficult, different in size, or that fill a particular niche. We also did not have usage data for GWT, due to the Google Code search service recently shutting down.

A final limitation of our study is that we obtained our results from Stack Overflow. The results may not extend to other social media, or similar sites reflecting different cultures or mediums. For example, the design and community associated with sites like Quora or Reddit may offer different dynamics than the ones we have observed. In addition, the question and answer format may constrain the nature of contributions when compared to other formats such as blogs. With blogs, the ability of going into deeper levels of discussion or offer opinions may reflect different content and dynamics than the ones we observed.

8. CONCLUSION

We have shown several sources of evidence that crowd documentation exists as a viable process that can emerge from social media sites, such as Stack Overflow, for creating software documentation. Documentation can emerge in the form of questions and answers that feature many code examples and discussions about using API classes and methods. The authors that contribute these items take distinct roles in curating and maintaining the quality of questions and answers.

Crowd documentation is an effort that is shared by many, and even if many only contribute a few items, the net result can achieve a high coverage of API functionality with a large impact reaching huge audiences. The process works with similar principles as open source development but is driven by unique and complex factors.

Finally, we highlighted tools and future directions for harnessing the efforts of the crowd in automatically improving API documentation, API analytics for API designers wishing to monitor the public reception of their creations, and guidelines for steering the crowd.

Acknowledgments

We wish to thank Fernando Figueira Filho, Cassandra Petrachenko, Peter C. Rigby and Jamie Starke for their feedback on earlier versions of this paper, and Patrick Gorman for contributions to the visualizations used in our study.

9. REFERENCES

[1] G. Butler, R. K. Keller, and H. Mili. A framework for framework documentation. *ACM Comput. Surv.*, 32, March 2000.

[2] P. Clements, D. Garlan, L. Bass, J. Stafford, R. Nord, J. Ivers, and R. Little. *Documenting Software Architectures: Views and Beyond*. Pearson Education, 2002.

[3] B. Dagenais and M. P. Robillard. Creating and evolving developer documentation: understanding the decisions of open source contributors. In *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, FSE '10, pages 127–136, New York, NY, USA, 2010. ACM.

[4] U. Dekel and J. D. Herbsleb. Improving api documentation usability with knowledge pushing. In *ICSE '09: Proceedings of the 31st International Conference on Software Engineering*, pages 320–330, Washington, DC, USA, 2009. IEEE Computer Society.

[5] A. Forward and T. C. Lethbridge. The relevance of software documentation, tools and technologies: a survey. In *DocEng '02: Proc. of the Symp. on Document engineering*, pages 26–33, New York, NY, USA, 2002. ACM.

[6] R. Holmes and G. C. Murphy. Using structural context to recommend source code examples. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 117–125, New York, NY, USA, 2005. ACM.

[7] H. C. Jiau and F.-P. Yang. Facing up to the inequality of crowdsourced api documentation. *SIGSOFT Softw. Eng. Notes*, 37(1):1–9, Jan. 2012.

[8] R. E. Johnson. Documenting frameworks using patterns. In *OOPSLA '92: conference proceedings on Object-oriented programming systems, languages, and applications*, pages 63–76, New York, NY, USA, 1992. ACM.

[9] W. L. Johnson and A. Erdem. Interactive explanation of software systems. In *Proceedings of The 10th Knowledge-Based Software Engineering Conference*, pages 155–164, Washington, DC, USA, 1995. IEEE Computer Society.

[10] J. Kim, S. Lee, S.-w. Hwang, and S. Kim. Adding examples into java documents. In *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering*, ASE '09, pages 540–544, Washington, DC, USA, 2009. IEEE Computer Society.

[11] T. C. Lethbridge, J. Singer, and A. Forward. How software engineers use documentation: The state of the practice. *IEEE Software*, 20:35–39, 2003.

[12] T. C. Lethbridge, J. Singer, and A. Forward. How software engineers use documentation: The state of the practice. *IEEE Softw.*, 20(6):35–39, 2003.

[13] 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 2011 annual conference on Human factors in computing systems*, CHI '11, pages 2857–2866, New York, NY, USA, 2011. ACM.

[14] D. Mandelin, L. Xu, R. Bodík, and D. Kimelman. Jungloid mining: helping to navigate the api jungle. In *PLDI '05: Proceedings of the 2005 ACM SIGPLAN conference on Programming language design and implementation*, pages 48–61, New York, NY, USA, 2005. ACM.

[15] A. Mockus, R. T. Fielding, and J. Herbsleb. A case study of open source software development: the apache server. In *Proceedings of the 22nd international conference on Software engineering*, ICSE '00, pages 263–272, New York, NY, USA, 2000. ACM.

[16] D. Pagano and W. Maalej. How do developers blog? an explorative study. In *Proceedings of the Eigth International Working Conference on Mining Software Repositories*, 2011.

[17] D. L. Parnas and P. C. Clements. A rational design process: How and why to fake it. *IEEE Trans. Softw. Eng.*, 12(2):251–257, 1986.

[18] C. Parnin and C. Treude. Measuring api documentation on the web. In *Proceedings of the 2nd international workshop on Web 2.0 for software engineering*, Web2SE '11, pages 25–30, New York, NY, USA, 2011. ACM.

[19] M. P. Robillard. What makes apis hard to learn? answers from developers. *IEEE Softw.*, 26:27–34, November 2009.

[20] J. Stylos. Msdn programming help resources survey. Personal Communication.

[21] C. Treude, O. Barzilay, and M.-A. Storey. How do programmers ask and answer questions on the web?: Nier track. In *Proceeding of the 33rd international conference on Software engineering*, ICSE '11, pages 804–807, New York, NY, USA, 2011. ACM.

[22] C. Treude and M.-A. Storey. Effective communication of software development knowledge through community portals. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, SIGSOFT/FSE '11, pages 91–101, New York, NY, USA, 2011. ACM.

[23] D. Čubranić and G. C. Murphy. Hipikat: recommending pertinent software development artifacts. In *ICSE '03: Proceedings of the 25th International Conference on Software Engineering*, pages 408–418, Washington, DC, USA, 2003. IEEE Computer Society.