API Documentation and Software Community Values: A Survey of Open-Source API Documentation

Robert Watson, Mark Stamnes, Jacob Jeannot-Schroeder, Jan H. Spyridakis
Department of Human-Centered Design & Engineering
University of Washington
Campus Box 352315, Seattle, WA 98195
01.206.685-1557
[rbwatson | mstamnes | jjs5 | jansp]@uw.edu

ABSTRACT

Studies of what software developers need from API documentation have reported consistent findings over the years; however, these studies all used similar methods—usually a form of observation or survey. Our study looks at API documentation as artifacts of the open-source software communities who produce them to study how documentation produced by the communities who use the software compares to past studies of what software developers want and need from API documentation. We reviewed API documentation from 33 of the most popular open-source software projects, assessed their documentation elements, and evaluated the quality of their visual design and writing. We found that the documentation we studied included most or all the documentation elements reported as desirable in earlier studies and in the process, we found that the design and writing quality of many documentation sets received considerable attention. Our findings reinforce the API requirements identified in the literature and suggest that the design and writing quality of the documentation are also critical API documentation requirements that warrant further study.

Categories and Subject Descriptors

H.5.2 [**User Interfaces**]: Training, help, and documentation

Keywords

API, API reference documentation, Application programming interface, Software documentation, Software libraries

1. INTRODUCTION

Application-programming interfaces (APIs) allow one program or web site to access the data and services provided by another program or website. APIs make programming easier by sharing code and enabling software reuse, and they are multiplying. Microsoft's .NET Framework grew from 35,470 API elements in 2002 to over 109,000 API elements in 2007 [1]. Since that report, Microsoft added several thousand API elements with Windows 8. Each month for the past few years, hundreds of APIs have also been added to the Programmable Web, a site that catalogs web-service APIs [2]. Each new API includes new features, which software developers must learn and apply quickly and correctly. This rapid growth shows no sign of abating, and the demand for increasingly short time-to-market puts tremendous pressure on today's software developers to learn and apply these new APIs.

While the surveys and interviews conducted in past studies of API-documentation requirements paint a consistent picture, recent studies suggest that API documentation might not provide software developers with what they need. One study found "that some of the most severe obstacles faced by developers learning new APIs pertained to the documentation and other learning resources" [3]. Lethbridge et al. [4] reported that documentation was "often poorly written" and "finding useful content in documentation can be so challenging that people might not try to do so." They found that "inline comments [in the source code] are often good enough to greatly assist detailed maintenance work." Another researcher described how API documentation was so bad that "developers may be getting as much as 50% of their documentation from Stack Overflow" (a web site that hosts questions and answers about software development) [5]. Looking deeper into these studies and reports reveals the diverse and complex nature of API documentation and its study.

To consider a different perspective from that of the past studies and to add some context to the recent observations, we look at what software development communities put into the API documentation they produce for themselves. We asked the research question: *do software development communities create documentation that contains, at a macro level, the documentation elements software developers have said they want in earlier literature?*

Past studies applied research methods in which the participants knew they were involved in the research and all produced very similar findings. Our study looks at the question from a different perspective, allowing us to triangulate the findings of past studies. In our study, we examine the documentation produced by the open-source software communities as artifacts of what developers value.

Open-source software is developed and supported by a community of individuals who create, use, and maintain the software and documentation. Therefore, the software and the documentation we find in open-source communities should represent what they value—that is, the members of a software community will tend to write only what they find valuable or useful (be it software or documentation). Because the community that forms around any individual piece of software is specific to that software, we studied a group of open-source software to obtain a more generalized sense of open-source software documentation.

SIGDOC'13, September 30–October 1, 2013, Greenville, North Carolina, USA.
Copyright © 2013 ACM 978-1-4503-2131-0/13/09…$15.00.
http://dx.doi.org/10.1145/2507065.2507076

2. BACKGROUND

Our study draws on past research in which software developers were observed, surveyed, and interviewed to identify the aspects of software documentation they need to do their job, or whose absence complicates it. From this research, we summarized the requirements of API documentation and evaluated the API documentation of open-source software.

2.1 Past Studies of Software Developers

Our list of elements that software developers require from documentation comes from past studies of software developers. Nykaza et al. [6] studied the installation of a customer-service call center and interviewed the software developers who used the system's SDK to write the software that adapted the system to the installation. Lethbridge et al. [4] studied software documentation used to maintain the software, as opposed to apply the software in another application. The scenario in Lethbridge et al. differs from that of API documentation written for an external audience in terms of purpose and audience, but includes many of the same requirements of API documentation for learning an API. Robillard surveyed [7] and later interviewed [3] a group of Microsoft software engineers to identify obstacles to learning a new API. Sillito and Begel [8] interviewed software developers at Microsoft about how they learned to develop software on a new software platform. Each of these studies listed some or all of the following API documentation elements as helpful or critical to learning an API.

- Overview documentation.
- Short code "snippets" that demonstrate usage of an API in context.
- Code examples that show best practices with an API.
- Scenario and task-based documentation.
- Limitations and error handling.
- *Meaningful* documentation (as opposed to "filler" or "boilerplate" content that adds little or no value to what is obvious).

Other studies of computer users, users who were not necessarily software developers per se, relate similar requirements of documentation [9] [10].

- Accuracy, completeness, and correctness.
- Scenario and task-focused examples.
- Content that does not repeat the obvious, such as what can be learned from the user interface.

Because software developers are computer users, we also considered those documentation requirements in our study.

2.2 Open-Source Software and Developers

A variety of research has focused on the motivations of software developers who contribute to open source software [13, 17, 18, 19, 20, 21, 22]. Hertel et al. [18] present high-level descriptions of their motivations, citing norm-oriented motives, pragmatic motives, hedonistic motives, and social/political motives, among others. Hars and Ou [17], drawing from psychological theories, distinguish between internal factors and external factors as motivations for contributing to open source projects. Internal factors include intrinsic motivation, referring to an "…innate desire to code," as well as altruistic motivations, and a sense of community identification. External factors include future rewards or the satisfaction of personal needs. Future rewards can include direct revenues from code or coding skills, knowledge gained from the coding experience that can be marketed, along with the self-marketing to potential employers, and just for the recognition from their peers in the community. The personal needs mentioned include the initiation of projects to create products to fill gaps in the current software by opening them to the community.

We believe that the motivation for documenting open source software corresponds to the motivations for developing for open source projects. Documenting open source software remains an important part of realizing the vision of the software developer or developers for the open source project. Oram [23] suggests several reasons why community documentation, that is, documentation generated by developers and individual users with the goal of helping others use the open source software, exists. Much like the explanations for motivations about creating open source software, documentation of open source software can be motivated by factors that are personal and for the betterment of the community who uses the software. The reasons for developing documentation include providing informal support outside of any official documentation to promote the software and helping others on the assumption that the documentation writer will be helped in the future. Helping others in the community also provides a sense of personal gratitude and builds a reputation amongst the open-source community, which can lead to personal growth for the writer. Oram [23] also points out that there are potential financial results for documentation through paid sites. While many of these motivations are external, we feel that the resulting documentation represents the values of the community because the individual members of the community decide to write it.

2.3 Evaluating API Documentation

We studied API documentation as an artifact of the software development communities that exists around open-source software libraries and applications. Studying artifacts is common in contextual design [11] and anthropological research [12]. Because these artifacts are produced by the community, they represent what the community values [13].

We used a heuristic evaluation method [14] to assess the artifacts we found in a way that would be consistent across all artifacts and enable us to study them individually and as a group. We collected the list of elements for our heuristic evaluation from the literature cited in the previous section and Watson [15], who summarized the high-level components of API documentation. Because we were reviewing online documentation only, we also referred to the Association of Support Professionals [16] best support site criteria document for additional insight into creating our list of evaluation criteria.

3. METHOD

We assessed API documentation of open-source software libraries for the presence of the documentation elements and the page design and writing quality. Based on the existing literature, we designed the method to test the hypothesis: *The documentation of open-source software will contain the elements that software developers want, as reported in past research.* To test this, we developed a list of documentation elements identified in past studies and then evaluated open-source API documentation sets, tabulating the documentation elements we found.

3.1 API Documentation Studied

We wanted to find a collection of software that represented a range of open-source software development communities. Our first attempt to select documentation for the study was to take a simple random sample from the catalog of more than ½ million open-source software apps and libraries listed at www.ohloh.net, a catalog of open-source software that has been used in other studies of open source software [24]. However, the vast majority of the software we found using this method had very small communities as measured by users and contributors listed in the catalog. Many of the projects from our initial random sample showed very little activity, appeared to have very few users, or did not appear to be viable projects. We decided that studying inactive or abandoned projects would not accurately reflect the values of an ongoing and active software community.

To make sure we studied viable software communities, we took another sample by selecting the 100 most popular applications listed on ohloh.net. Studying the most popular applications would allow us to study the artifacts of a software community that had enough time and resources to enable the documentation to reach a state that represents the community's values. While this sample does not represent all the software in the open-source catalog, it does represent the more active software projects in the catalog—those that have a large number of the open-source software developers who are the ultimate subject of our research.

Of the 100 open-source projects we started with, we eliminated the projects that did not have a programmable interface intended for software developers. Some of the projects we studied had both an end-user interface and an API for software developers. In those cases, we studied only the API. We also eliminated command-line tools, operating systems, and system-building projects because they represent niche audiences that are distinct from those of general APIs. The result was the 33 open-source software projects listed in Table 1.

Table 1. Open-source API documentation studied

- Adblock Plus
- Apache OpenOffice
- CakePHP
- Common Unix Printing System (CUPS)
- Cygwin
- Django
- Drupal (core)
- Eclipse Platform Project
- FileZilla Firebug
- GIMP
- Git
- GNOME
- GTK+
- Hibernate
- Inkscape
- jQuery
- JUnit
- KDE
- MediaWiki
- MySQL
- NetBeans IDE
- Perl
- PHP
- PostgreSQL Database Server
- Python programming language
- Ruby on Rails
- Samba
- SQLite
- Subversion
- Trac
- VirtualBox-Open Source Edition
- WordPress

3.2 Study Heuristic

We grouped the API documentation elements into three general categories for our evaluation:

- **Overall documentation elements**
  Elements that characterize the general nature of the developer documentation.
- **Documentation entry/home page elements**
  Elements found on the "home page" or top-level page of the developer documentation.
- **API reference topic elements**
  Elements found in the API reference topics.

3.2.1 Assessment elements

Tables 2, 3, and 4 list the specific documentation elements we assessed in each category.

Table 2. Overall documentation aspects

Question	Rating Scale
How did you find the documentation? (the navigation method used)	Link from home page Link from other page Search, internal to the site Search, external to the site
Can you find video tutorials for using the API in the documentation?	Yes No
Can you find sample apps or links to samples in the documentation?	Yes No
*Provide a qualitative estimate of the site quality as a whole.	Excellent Good Fair Poor Terrible Other

How easy was it to find the documentation?	Easy = effortless
	Hard = not easy

Can you find code tutorials in the documentation?	Yes No
Can you find an API Overview in the documentation?	Yes No
Note any comments from your experience with the site.	Free text comment field

The element noted by an asterisk in Table 2 was reviewed separately from the elements in Table 5. The evaluation of "Provide a qualitative estimate of the site quality as a whole" occurred during the initial evaluation of the documentation to capture a "first impression" of the documentation. After reviewing the ratings of the site quality overall, we added the criteria in Table 5 to identify some of the components that might have contributed to the first-impression rating. The elements in Table 5 were then reviewed in a second evaluation.

Table 3. Entry page documentation elements

Question	Rating Scale
Note the entry page URL	URL of page
Does the entry page have a documentation overview or a link to an overview?	Yes No
Does the entry page have a value proposition for the API?	Yes No
Does the entry page have getting-started content or a link to getting-started content?	Yes No
Does the entry page have a table-of-contents?	Yes No
Note any other comments from your experience with the entry page.	Free text comment field

Table 4. API reference topic elements

Question	Rating Scale
How did you find the API Reference?	Link from home page Link from other page Search, internal to the site Search, external to the site
Note the API reference topic homepage URL	URL of page
Describe the navigation used by the reference topics	Hub-Spoke Menu-Content Other
Did the reference topics provide interactions with the code?	Yes No

How easy was it to find the API reference?	Easy = effortless
	Hard = not easy

Estimate the API Size (from the number of ref. topics).	Small: APIs with < 10 high-level objects (e.g. classes, objects, etc.) Medium: APIs with 10-99 high-level objects Large: APIs with 100-999 high-level objects Huge: APIs with 1,000 or more high-level objects
Describe how the reference topic pages are organized (multi/single).	Single-Elem/Page Multiple-Elem/Page Other
Did you find code snippets in most of the reference topics you studied?	Yes No

Table 5. Design and writing quality criteria

Question	Rating Scale
Rate the level of design elements used on a reference topic.	High: Many design elements, such as multiple fonts, text layout styles, images, and other visual design elements such as lines, shadings, and adaptive page design. Lo: One or two fonts, minimal use of layout and visual design elements such as lines and shading.
Rate the reference topic pages' content quality in terms of richness and clarity.	High: writing is clear, detailed, and can be understood, even by someone who is not familiar with the API. Lo: writing is unclear, lacking in detail, and is difficult to understand.

The rating scales for the questions in Table 5 were intentionally general to make them easy to rate consistently while providing enough detail to identify the patterns and sites that might merit further study.

3.2.2 Documentation Elements from Past Research

Table 6 shows how the elements we summarized from past research match the assessment elements in our study.

Table 6. Past research and assessment elements

Documentation element from past research	Assessment questions in this study
Overview documentation.	Does the entry page have a documentation overview or a link to an overview? Can you find an API Overview in the documentation?
Short code "snippets" that demonstrate usage of an API in context.	Did you find code snippets in most of the reference topics you studied?
Code examples that show best-practices with an API	Can you find sample apps or links to samples in the documentation?
Scenario and task-based documentation.	Can you find code tutorials in the documentation
Limitations and error handling.	Not studied
Meaningful documentation (as opposed to "filler" or "boilerplate" content that adds little or no value to what is obvious).	Rate the reference topic pages' content in terms of richness and clarity.

The limitations and error-handling element was not rated in this study because we could not characterize it in a way that we could evaluate.

3.3 Study Method and Coding

Four researchers evaluated the API documentation of the selected software projects (Table 1) for the elements listed in the preceding section. Three of the four researchers had used APIs and API documentation in the past to develop software.

The researchers practiced coding API documentation that was not part of the study sample to improve inter-rater reliability and refine the definitions of the documentation elements. Through multiple iterations and review sessions, the researchers determined the operational definitions of each element in the evaluation rubric. At least one coder then assessed the documentation of each API in the final set of APIs and a subset of the APIs was reviewed by the other coders for consistency and agreement. The few disagreements found in this process were reviewed and resolved by agreement of all researchers before the data were analyzed.

4. FINDINGS

We evaluated the API documentation of the open-source software listed in Table 1 to find the documentation elements listed in Tables 2-4. In the first evaluation, we tabulated the characteristics described in our rubric; however, we also identified aspects of the documentation that the original survey did not include. We added the elements in Table 5 to our rubric and then evaluated those elements of the documentation.

4.1 Documentation Elements

Table 7 lists the frequency of the key documentation elements in the API documentation studied. Except for the API overviews, our findings support our hypothesis in that the documentation elements listed as required or desired by software developers in API documentation were found in most of the API documentation we studied.

Table 7. Key documentation elements in documentation studied (n = 33)

Documentation element evaluation question	
Does the entry page have a documentation overview?	82% Yes
Can you find an API Overview in the documentation?	42% Yes
Did you find code snippets in most of the reference topics you studied?	85% Yes
Can you find code tutorials in the documentation	79% Yes
Can you find sample apps or links to samples in the documentation?	55% Yes
Rate the reference topic pages' content in terms of richness and clarity.	82% Good or Exc.

4.2 Design and Writing Quality Evaluation

In our first evaluation of the documentation elements, we found that 21 of the 33 documentation sets (64%) had an overall impression of good or excellent; however, we encountered a broad range graphic-design and writing styles. To characterize this variation better, we added the evaluation criteria listed in Table 5 and reviewed the documentation sets again to evaluate these elements. We found that the qualitative ratings were surprisingly high—specifically, more than half of the open-source documentation we studied (19 of 33) had both high-quality design and high-quality writing (Table 8). This supports the notion that the software development communities value quality in both design and writing, which suggests that craftsmanship is also valued. High-quality writing appeared more often than high-quality design—we found that 82% of the API documentation studied had high-quality writing as compared to the 61% that had high-quality design. Table 8 shows the results of this evaluation. Using a Pearson Chi-Square test, we found significant patterns in the design and writing quality, revealing that writing quality was high in most cases.

Table 8. Design and writing

		Writing Quality		Pearson Chi-Square
		Low (n=6)	High (n=27)	
Design Quality	Low (n=13)	5	8	χ^2 (1, N = 33) = 5.93, p = .015
	High (n=20)	1	19	

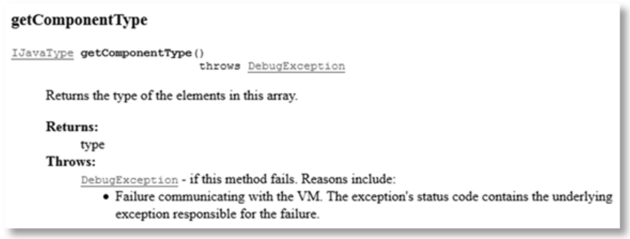
The following sections illustrate examples of the different types of design and writing we encountered in our study.

4.2.1 Low-Design/Low-Writing Quality

We grouped documentation sets into the *low-design/low-writing quality* group if the reference pages had:

- A page design with only one or two fonts, minimal use of layout and visual design elements such as lines and shading.
- Page content where the writing is unclear, lacking in detail, or is difficult to understand.

Figure 1 [25] is an example of a reference topic with minimal visual design elements and writing that provides very little detail.



getComponentType

```
IJavaType getComponentType()
                           throws DebugException
```

Returns the type of the elements in this array.

Returns:
type
Throws:
DebugException - if this method fails. Reasons include:

- Failure communicating with the VM. The exception's status code contains the underlying exception responsible for the failure.

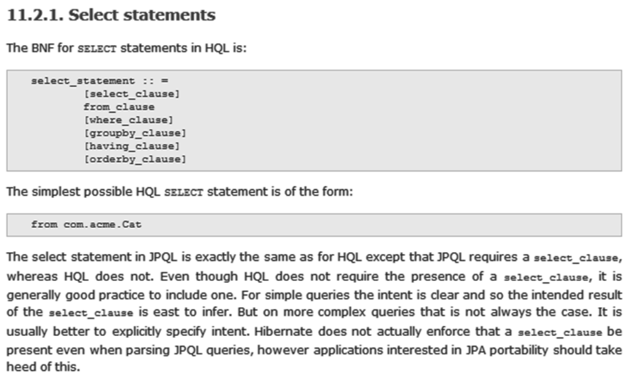
Figure 1. Example of low-design/low-writing qualitydocumentation Copyright (c) 2000, 2007 IBM Corporation and others.

4.2.2 Low-Design/High-Writing Quality

We grouped documentation sets into the *low-design/high-writing quality* group if the reference pages had:

- A page design with only one or two fonts, minimal use of layout and visual design elements such as lines and shading.
- Page content where the writing is clear, detailed, and can be understood, even by someone who is not familiar with the API.

Figure 2 [26] is an example of a reference topic with minimal visual design elements, but detailed text.



11.2.1. Select statements

The BNF for `SELECT` statements in HQL is:

```
select_statement :: =
        [select_clause]
        from_clause
        [where_clause]
        [groupby_clause]
        [having_clause]
        [orderby_clause]
```

The simplest possible HQL `SELECT` statement is of the form:

```
from com.acme.Cat
```

The select statement in JPQL is exactly the same as for HQL except that JPQL requires a `select_clause`, whereas HQL does not. Even though HQL does not require the presence of a `select_clause`, it is generally good practice to include one. For simple queries the intent is clear and so the intended result of the `select_clause` is east to infer. But on more complex queries that is not always the case. It is usually better to explicitly specify intent. Hibernate does not actually enforce that a `select_clause` be present even when parsing JPQL queries, however applications interested in JPA portability should take heed of this.

Figure 2. Example of low-design/high-writing quality documentation

4.2.3 High-Design/Low-Writing Quality

We grouped documentation sets into the *high-design/low-writing quality* group if the reference page had:

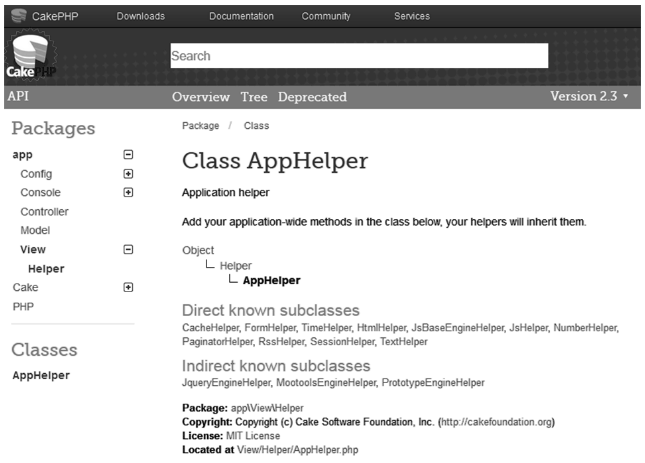
- A page design with many design elements, such as multiple fonts, text layout styles, images, and other visual design elements such as lines, shadings, and adaptive page design.
- Page content where the writing is unclear, lacking in detail, or is difficult to understand.

Figure 3 [27] is an example a reference topic with many visual design elements, but writing that lacks detail.

4.2.4 High-Design/High-Writing Quality

We grouped documentation sets into the *high-design/high-writing quality* group if the reference page had:

- A page design with many design elements, such as multiple fonts, text layout styles, images, and other visual design elements such as lines, shadings, and adaptive page design.
- Page content where the writing is clear, detailed, and can be understood, even by someone who is not familiar with the API.



Class AppHelper

Application helper

Add your application-wide methods in the class below, your helpers will inherit them.

```
Object
  └ Helper
      └ AppHelper
```

Direct known subclasses

CacheHelper, FormHelper, TimeHelper, HtmlHelper, JsBaseEngineHelper, JsHelper, NumberHelper, PaginatorHelper, RssHelper, SessionHelper, TextHelper

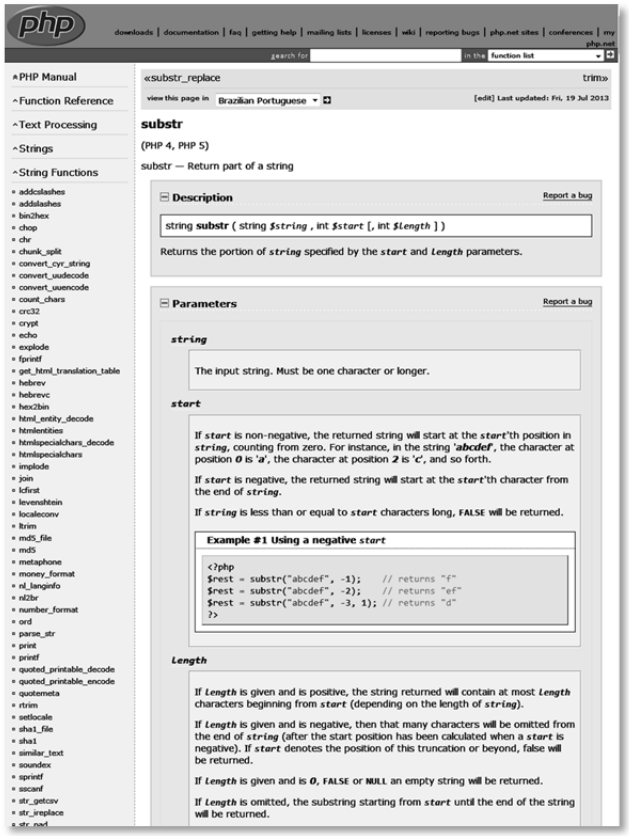
Indirect known subclasses

JqueryEngineHelper, MootoolsEngineHelper, PrototypeEngineHelper

Package: app\View\Helper
Copyright: Copyright (c) Cake Software Foundation, Inc. (http://cakefoundation.org)
License: MIT License
Located at View/Helper/AppHelper.php

Figure 3. Example of high-design/low-writing quality documentation. Copyright © 2013 Cake Software Foundation, Inc.

Figure 4 [28] is an example of a reference topic with numerous visual styling elements and detailed text.



[edit] Last updated: Fri, 19 Jul 2013

substr

(PHP 4, PHP 5)

substr — Return part of a string

Description

string **substr** (string *$string* , int *$start* [, int *$length*])

Returns the portion of *string* specified by the *start* and *length* parameters.

Parameters

string

The input string. Must be one character or longer.

start

If *start* is non-negative, the returned string will start at the *start*'th position in *string*, counting from zero. For instance, in the string '*abcdef*', the character at position *0* is '*a*', the character at position *2* is '*c*', and so forth.

If *start* is negative, the returned string will start at the *start*'th character from the end of *string*.

If *string* is less than or equal to *start* characters long, FALSE will be returned.

Example #1 Using a negative *start*

```
<?php
$rest = substr("abcdef", -1);    // returns "f"
$rest = substr("abcdef", -2);    // returns "ef"
$rest = substr("abcdef", -3, 1); // returns "d"
?>
```

length

If *length* is given and is positive, the string returned will contain at most *length* characters beginning from *start* (depending on the length of *string*).

If *length* is given and is negative, then that many characters will be omitted from the end of *string* (after the start position has been calculated when a *start* is negative). If *start* denotes the position of this truncation or beyond, false will be returned.

If *length* is given and is *0*, FALSE or NULL an empty string will be returned.

If *length* is omitted, the substring starting from *start* until the end of the string will be returned.

Figure 4. Example of high-design/high-writing quality documentation. Copyright © 2001-2013 The PHP Group.

4.3 Analysis of Design and Writing Quality

To test for consistency between the detailed quality elements and our initial overall assessment, we compared our initial overall quality assessment to the more specific ratings recorded later in the study by using a Pearson Chi-Square test. Table 9 shows that we found a statistically significant ($p < .05$) relationship between the overall rating and the specific ratings, indicating consistency between them.

We looked for a significant relationship between the frequency of the document elements listed in Table 6 and the quality evaluations using three one-way ANOVA tests. Table 10 shows that the average frequency of the documentation elements found in an API documentation set increased with the assessments of the design and writing quality, as well as the overall impression of the documentation set.

While documentation with high quality writing, design, and with good and excellent overall site quality had more elements then low quality documentation, on average, there was a significant effect seen at the $p < .05$ level in only the writing quality and site quality on the number of documentation elements found. There was no significant effect observed between the element frequency and the design quality.

Table 9. Overall impression and quality dimensions

		Overall evaluation of API documentation		
		Poor - Fair	Good - Exc.	**Pearson Chi-Square**
Design Quality	Low	10	3	χ^2(1, N = 33) = 15.249, *p* = .000
	High	2	18	
Writing Quality	Low	5	1	χ^2(1, N = 33) = 6.991, *p* = .008
	High	7	20	

Table 10. Document element frequency and quality

		Number of document element categories found			
		Mean	**Std. Dev.**	**N**	**ANOVA**
Design Quality	**Low**	3.31	1.601	13	[F (1,31) = 1.877, *p* = 0.181]
	High	4.05	1.468	20	
Writing Quality	**Low**	2.33	1.366	6	[F (1,31) = 7.537, *p* = 0.010]
	High	4.07	1.412	27	
Site Quality	**Fair - Poor**	2.92	1.443	12	[F (1,31) = 6.590, *p* = 0.015]
	Good - Exc.	4.24	1.411	21	

To see which of the individual elements, if any, were associated with high-quality documentation, we looked at the frequency of each of the documentation elements listed in Tables 2, 3, and 4. To find a significant relationship between each one and the quality factors, we used a Pearson Chi-Square test. Tables 11 and 12 show the only significant patterns we found. Of all the documentation elements we evaluated, only the presence of code tutorials and code snippets showed a significant relationship ($p < .05$) with design and writing quality.

Table 11. Code tutorials and quality

		API documentation has code tutorials		Pearson Chi-Square
		No	Yes	
Design Quality	Low	5	8	χ^2 (1, N = 33) = 3.82, *p* = .051
	High	2	18	
Writing Quality	Low	4	2	χ^2 (1, N = 33) = 9.07, *p* = .003
	High	3	24	

Table 12. Code snippets and quality

		API reference topics have code snippets		Pearson Chi-Square
		No	Yes	
Design Quality	Low	4	9	χ^2 (1, N = 33) = 4.07, *p* = .044
	High	1	19	
Writing Quality	Low	3	3	χ^2 (1, N = 33) = 6.93, *p* = .008
	High	2	25	

5. DISCUSSION

The presence of most of the key elements in the documentation we studied supports the hypothesis that open-source software development communities, at a macro level, put the same documentation elements into their documentation as the software developers asked for in studies and interviews. That the software communities voluntarily include these elements in the documentation they produce supports the idea that they value these API documentation elements whether they are responding to surveys and interviews, or actually writing software and documentation.

In the course of this study, we experienced several occasions in which we needed to review and revise our evaluation criteria. Initially, we required several rounds of practice evaluations to refine the operational definitions of the elements we were studying. After our first full review, we found high variability in the perceived quality of the API documentation, even though we were studying the most popular open-source software products listed in ohloh.com. To investigate this variation further, we evaluated the design and written quality of the documentation sets in a second review of the API documentation.

5.1 Dealing with the Diversity

Early in the study, we had difficulty identifying some of the documentation elements with consistency using our initial definitions because they did not accommodate the variety of documentation we encountered. While the documentation element definitions seemed clear at the beginning of the project, as we applied them in our initial assessment, it became evident they needed refinement to accommodate the diversity of documentation styles, page formats, and rater experience levels. It took several iterations of trial-and-review to refine the operational definitions of the different elements such that the reviewers could assess the documentation in a reliable, consistent way.

The wide variety of content we encountered and the difficulty we had finding and applying a consistent definition of those elements makes us wonder about the consistency of what past researchers have studied and how we talk about these elements in the literature. In this study, we found that some documentation elements were easier to recognize than others were. For example, Robillard [3] describes a taxonomy of program-code examples that was clear and easy for the researchers to recognize.

- Code-snippet (showing the function being called in a specific context for illustration).
- Sequences of small examples to illustrate functionality (tutorial examples).
- Sample apps (complete and functional programs that use the function).
- Production code (source code of software that is uses the function in a customer-facing application or scenario).

On the other hand, identifying the elements that made up the intent documentation [3] was more challenging. While we tried to operationalize this in a way that mapped to recognizable documentation elements, some intent documentation, such as that which describes specific performance, usage limitations, or error conditions, might be found inside specific reference topics and not in a single topic for an API. Intent documentation that describes higher-level concepts of how to use the API in context, on the other hand, might be more appropriate in an overview or some other type of conceptual topic that focuses on the API as a whole rather than just a single element of the API. Having the information distributed around the documentation could make it hard for developers to know where to find such information, or to know if it even exists at all, until they spend time learning what is and is not documented. It also makes it hard to assess its presence with any accuracy in a survey such as this one.

In addition to the different forms the elements sought by developers can take, the vocabulary used is also critical to discovery. Ko and Riche [29], observed how documentation could exist but remain invisible to the user if they did not know the correct vocabulary to use to find it. Finally, Robillard [3] points out that too much intent documentation can make the documentation difficult to use suggesting that this might be judged better in a specific context, rather than just a simple test for presence.

The diversity of API documentation content and format presented a challenge to our study. On the one hand, technical writing curriculum tells technical writers to know their audience and to write to them [30]. Given that the documentation we studied was written by the community who also use it, it is reasonable to assume they are writing the documentation they need in the format they prefer. Such a focused approach makes sense in a context limited to a single API or library. However, today, it is increasingly common for software developers to use software and documentation from a wide variety of sources as they adopt and apply new technologies. Documentation written to a specific audience can present challenges for developers who come from another perspective or background.

While our survey identified the presence of specific documentation elements, it did not address the usability of the content or its suitability to any task. The elements we studied were presumed to be useful in that they had been identified as necessary or desired in the literature. The diversity of the documentation we reviewed, however, indicates that the problem might be more nuanced than just ensuring the API documentation has a collection of requested elements, for example the matter of craftsmanship in the documentation design and writing.

5.2 Recognizing Craftsmanship

In spite of the diverse character of the API documentation we reviewed, high quality writing and attention to detail was more common than not. The high percentage of documentation sets we found with writing that was clear and detailed indicates quality writing is a common value—one that has not been discussed much in the literature about API documentation. That there was documentation without it, however, indicates that it is a property that cannot be assumed. The use of visual design elements in the API reference topics was also higher than we expected at the beginning of the study, and it, too, is an aspect of API documentation that has not received much coverage.

While the occurrence of high quality design and writing suggests that attention to detail in the content is valued, we do not know if it has any effect on usability or popularity. In our study, for example, we found no significant relationship between any of the quality dimensions and the software's popularity rank or rating in ohloh.com. All the API documentation we studied came from the top 100 open-source software projects in ohloh.net, yet there was considerable variation in the documentation quality.

5.3 Threats to Validity

We assessed the API documentation of open-source software for the presence of specific documentation elements. In any specific documentation sample, the documentation could have been produced by the community on demand, by an organized or professional documentation effort, or some combination—the extent of which we do not know. As open-source software and documentation, we assume that the API documentation represents the values and needs of each individual community.

As open-source software and documentation, it is possible; in fact, it is quite likely, that the API documentation we studied does not represent API documentation as a whole. However, that is not relevant to the research question of the study because we are using open-source and community-generated documentation as an artifact of the software community to gain insight into what they value. In that regard, a selection of open-source and community-driven documentation is appropriate. The literature indicates that there is a high-degree of overlap between open-source and professional, commercial software developers. In many cases, they are the same people. As such, these findings should represent their values whether they are programming for hire or not. While we feel that our findings reflect the values of software developers, these findings should not be generalized to API documentation that was not included in the study.

One aspect of the open-source documentation that became known in our assessment was that many examples of open-source software documentation relied on content that was not part of an organized documentation set. Open-source software developers (if not *all* software developers) are accustomed to using community content such as forum posts, blogs, and other unstructured documentation. While not recorded in our findings, we observed that relying on these unstructured types of documentation appeared to be common. This could, however, cause our study to understate the frequency of documentation elements for a specific API documentation set. For example, it might be common in a particular software community to have code samples or sample programs separate from the API documentation—in which case they would exist for and be known by the community, but could have been missed by our assessment method.

While we made some qualitative assessments of the documentation's visual design and writing quality, this analysis was conducted at a macro level. We did not perform any formal content analysis on the documentation sets we studied. Such analysis exceeded the scope of this research; however, the findings from this study suggest such an analysis would be worthwhile.

6. CONCLUSIONS

The findings of our study of API documentation as an artifact of open-source software communities corroborate the findings of past research into what software developers want and need in API documentation. Past research describes a need for the following elements in API documentation, most of which we found in the documentation we studied.

- Overview documentation.
- Short code "snippets" that demonstrate usage of an API in context.
- Code examples that show best-practices with an API
- Scenario and task-based documentation.
- Meaningful documentation (as opposed to "filler" or "boilerplate" content that adds little or no value to what is obvious).

The fact that most of the communities who support the software we studied provided these elements in the APIs they supported suggests they represent common values among open-source software developers. That the design quality and writing quality of the API documentation we studied were high indicates the developers in these software communities also value these attributes enough to include them in their documentation.

6.1 Documentation is more than the Sum of its Parts

Our study found that open-source software documentation has, for the most part, the elements that the literature identified as necessary. However, we also found that an inventory is not sufficient to characterize a documentation set accurately. Aspects such as design quality, writing quality, terminology, and navigational affordances are also critical elements to consider. While design and writing quality, per se, do not appear as requirements in the literature, the variation of these dimensions that we encountered suggests that the perceived need for such quality varies. Perhaps high-quality design and writing is assumed; however, our study indicates that high quality design and writing is not universally consistent. The variation that we found in these quality dimensions suggests there would be value into further study into how they affect API documentation usability and utility.

6.2 What are we Talking About, Anyway?

While answering some questions, our research also raises others. If the software development communities are producing the documentation that software developers are asking for when surveyed and interviewed, what is the basis of their recent complaints? This study gave us a new appreciation for the level of diversity that exists in the world of "API documentation." Our survey spanned a wide swath of API documentation, much wider than most software developers would tend to encounter in a similar period. At the same time, we touched upon only a very small part of the API-documentation universe. It is possible that past API studies and criticisms are each seeing small and different pieces of a much larger whole—not unlike the fable of the blind men describing an elephant.

6.2.1 Different Worlds

The literature we reviewed indicates that open-source software developers have a lot in common with professional (paid) software developers—in fact, the same people often work on both types of software. It is possible that the findings from our study of open-source API documentation do not generalize to the Microsoft developers' experience in Robillard [7] and Robillard and DeLine [3] or the Android developers' experiences in Parnin [5]. It could be that it is specific examples from these environments that do not meet the needs of the developers—not a general problem with overall documentation that is responsible for the findings in those studies. Further, the difficulty we experienced in operationalizing the element definitions at the beginning of our research suggests that differences in element definitions could be complicating the discussion. While there are differences in in the subjects of each study, the important point to remember is the agreement between their conclusions. However, the definitions and descriptions of the different elements need additional refinement and clarification for practitioners to be able to apply the findings from these studies.

6.2.2 Different Methods

Our study differs from those in the literature we studied in that we studied the products of software communities without involving them directly. We looked at the artifacts that result from their actions, without them knowing we were studying them—in fact, we looked at their work long after they completed it. In that sense, there was no way for our research to influence their actions. On the other hand, in most of the earlier studies, the researchers interacted directly with the participants—the developers. Robillard and DeLine's study [3] focused on learning obstacles by asking, for example, "For each type of obstacle described below, please rate how severe this type of obstacle was in your experience learning the API you mentioned above." Such a leading question could influence the response. While that research focuses on a single aspect of learning APIs, it also highlights the need, in a subject as large and diverse as this one, for multiple studies and multiple study methods to construct a complete picture. Any one method, by itself, is likely to tell only a partial story, at best.

6.2.3 Different Perspectives

While studying the elements of a documentation set provides an inventory of its contents, it does not describe the suitability for a specific task to a specific audience. For the target audience of software developers, the suitability of the documentation to their task is very relevant and likely to influence their opinion of a documentation set. To the software developers using the documentation, if they cannot find what they are looking for, to them, it does not exist—even if we found it in our inventory. This difference does not make an inventory less valuable; however, it might identify ways in which the inventory could be improved. While the different methods provide useful and different insights, it is especially important to recognize when the methods reinforce each other, for example, how our study reinforced the findings from the more direct study methods used to observe software developers in past studies.

7. FUTURE WORK

The diversity of documentation content and content styles we found identified more questions and opportunities for study. It would be valuable to know if variations of these aspects of the documentation influence the software developer's experience and assessment of it. Exploring the influence of these factors, for example, could inform future authoring systems and documentation templates to help make it faster and easier to produce API documentation that software developers need. Our study also reinforced the need to study the documentation in context, so identifying methods and practices to collect and report this information could help identify how to improve existing and plan future documentation.

Because the documentation we studied came from the community it serves, it is reasonable to assume that each specific community tailors the documentation for that community. In that sense, the diverse content we found in our study is a good thing. At the same time, such an uneven content landscape presented a challenge to us as researchers and we suspect that it also presents a challenge to software developers who have to work with multiple libraries and products. It would be helpful to know the impact of variations in content, layout, and organization on search, comprehension, and usability. We suspect these differences complicate developers' understanding and learning in using different APIs. As APIs and API users become more numerous and more diverse, this diversity could add documentation requirements that did not heretofore exist.

Modern software development also appears to encourage just-in-time learning for specific tasks and in moving on to the next task [8]. In such a scenario, software developers coming to any documentation set will arrive with the perspective of a new user more so than that of an expert. Although they might have used the software and documentation in the past, when they return after using other software, they will still need to familiarize themselves with the navigation and terminology all over again, just like a new user. Accommodating these scenarios, which were uncommon in the past, will require additional research to identify the best practices that will empower the users of API documentation today and into the future.

8. ACKNOWLEDGEMENTS

We would like to thank Black Duck Software who hosts OhLoh.com, which provided the catalog open-source software and the data for the software and documentation we studied.

9. REFERENCES

[1] Abrams, B. 2008. *Number of types in the .NET Framework*. Brad Abrams. Retrieved from http://blogs.msdn.com/b/brada/archive/2008/03/17/number-of-types-in-the-net-framework.aspx

[2] Programmable web. 2013. *Keeping you up to date with APIs, mashups and the Web as platfor*m. Retrieved May 19, 2013, from http://www.programmableweb.com/

[3] Robillard, M. P., & DeLine, R. 2011. A field study of API learning obstacles. *Empirical Software Engineering*, 16(6), 703–732. doi:DOI 10.1007/s10664-010-9150-8

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

[5] Parnin, C. 2013. *API documentation: Why it sucks. ninlabs research*. Retrieved May 19, 2013, from http://blog.ninlabs.com/2013/03/api-documentation/

[6] Nykaza, J., Messinger, R., Boehme, F., Norman, C. L., Mace, M., & Gordon, M. 2002. What programmers really want: Results of a needs assessment for sdk documentation. In *Proceedings of the 20th Annual International Conference on Computer Documentation* (pp. 133–141). Presented at the SIGDOC 2002, Toronto, Ontario, Canada: ACM.

[7] Robillard, M. P. 2009. What makes APIs hard to learn? answers from developers. *Software, IEEE*, 26(6), 27–34.

[8] Sillito, J., & Begel, A. 2013. App-directed learning: An exploratory study. Presented at the 6th International Workshop on Cooperative and Human Aspects of Software Engineering, San Francisco, CA, USA.

[9] Mitchell, G. E. 1993. What do users really want from computer documentation? In *IPCC 93 Proceedings. The New Face of Technical Communication: People, Processes, Products* (pp. 27–31). Presented at the International Professional Communication Conference 1993, IEEE.

[10] Novick, D. G., & Ward, K. 2006. What users say they want in documentation. Presented at the SIGDOC 2006, Myrtle Beach, South Carolina, USA: ACM.

[11] Holtzblatt, K., Wendell, J. B., & Wood, S. 2004. *Rapid contextual design: a how-to guide to key techniques for user-centered design*. Morgan Kaufmann.

[12] Bernard, H. 2006. *Research methods in anthropology: qualitative and quantitative approaches* (Fourth Edition.). Lanham, MD, USA: Altamira Press.

[13] von Krogh, G., Haefliger, S., Spaeth, S., & Wallin, M. W. 2012. Carrots and rainbows: Motivation and social practice in open source software development. *MIS Quarterly-Management Information Systems*, 36(2), 649–676.

[14] Nielsen, J. 1993. *Usability Engineering*. Boston. MA. USA: Academic press.

[15] Watson, R. B. 2012. Development and application of a heuristic to assess trends in API documentation. In *Proceedings of the 30th ACM International Conference on Design of Communication* (pp. 295–302). Presented at the SIGDOC 2012, Seattle, WA, USA: ACM.

[16] Association of Support Professionals, The. 2013. *The Ten Best Web Support Sites of 2013: Site Scoring System*. ASPonline.com. Retrieved May 19, 2013, from http://www.asponline.com/scoring13.pdf

[17] Hars, A., & Ou, S. 2002. Working for free? Motivations for participating in open-source projects. *International Journal of Electronic Commerce*, 6, 25–40.

[18] Hertel, G., Niedner, S., & Herrmann, S. 2003. Motivation of software developers in Open Source projects: an Internet-based survey of contributors to the Linux kernel. *Research policy*, 32(7), 1159–1177.

[19] Oreg, S., & Nov, O. 2008. Exploring motivations for contributing to open source initiatives: The roles of contribution context and personal values. *Computers in Human Behavior*, 24(5), 2055–2073. doi:doi:10.1016/j.chb.2007.09.007

[20] Shah, S. K. 2006. Motivation, governance, and the viability of hybrid forms in open source software development. *Management Science*, 52(7), 1000–1014.

[21] Wu, C.-G., Gerlach, J. H., & Young, C. E. 2007. An empirical analysis of open source software developers' motivations and continuance intentions. *Information & Management*, 44(3), 253–262. doi:doi:10.1016/j.im.2006.12.006

[22] Ye, Y., & Kishida, K. 2003. Toward an understanding of the motivation of open source software developers. In *Software Engineering, 2003, Proceedings. 25th International Conference on* (pp. 419–429). Presented at the Software Engineering, 2003, International Conference on, IEEE.

[23] Oram, A. 2007. Why do people write free documentation? results of a survey. LAMP: *The Open-Source Web Platform*. Retrieved from http://www.onlamp.com/lpt/a/7062

[24] Ellis, H. J., Purcell, M., & Hislop, G. W. 2012. An approach for evaluating FOSS projects for student participation. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education* (pp. 415–420). Presented at the SIGCSE 2012, Raleigh, North Carolina, USA.

[25] Help - Eclipse Platform. (n.d.). Retrieved July 20, 2013, from http://help.eclipse.org/juno/index.jsp?topic=%2Forg.eclipse.jdt.doc.isv%2Freference%2Fapi%2Forg%2Feclipse%2Fjdt%2Fdebug%2Fcore%2FIJavaArrayType.html&anchor=getComponentType(). Included under Eclipse Public License

[26] Chapter 11. HQL and JPQL. (n.d.). Retrieved July 20, 2013, from http://docs.jboss.org/hibernate/orm/4.1/devguide/en-US/html/ch11.html#d5e2552. Included under LGPL v2.1.

[27] Class AppHelper | CakePHP. (n.d.). Retrieved July 20, 2013, from http://api.cakephp.org/2.3/class-AppHelper.html. Copyright © 2013 Cake Software Foundation, Inc.

[28] PHP: substr - Manual. (n.d.). Retrieved July 20, 2013, from http://php.net/manual/en/function.substr.php. Copyright © 2001-2013 The PHP Group.

[29] Ko, A. J., & Riche, Y. 2011. The role of conceptual knowledge in API usability. In *Proceedings of the Visual Languages and Human-Centric Computing (VL/HCC), 2011 IEEE Symposium on* (pp. 173–176). Presented at the Visual Languages and Human-Centric Computing (VL/HCC), 2011 IEEE Symposium on, IEEE.

[30] Markel, M. 2006. *Technical Communication* (8th ed.). Boston. MA. USA: Bedford/St. Martins.