Application Programming Interface Documentation: What Do Software Developers Want?

Article *in* Journal of Technical Writing and Communication · July 2018

DOI: 10.1177/0047281617721853

CITATIONS
9

READS
3,523

3 authors, including:

Michael Meng
Hochschule Merseburg
30 PUBLICATIONS **519** CITATIONS

SEE PROFILE

Article

Application Programming Interface Documentation: What Do Software Developers Want?

Journal of Technical Writing and Communication
2018, Vol. 48(3) 295–330

DOI: 10.1177/0047281617721853
journals.sagepub.com/home/jtw
SAGE

Michael Meng[1], Stephanie Steinhardt[1], and Andreas Schubert[1]

Abstract

The success of an application programming interface (API) crucially depends on how well its documentation meets the information needs of software developers. Previous research suggests that these information needs have not been sufficiently understood. This article presents the results of a series of semistructured interviews and a follow-up questionnaire conducted to explore the learning goals and learning strategies of software developers, the information resources they turn to and the quality criteria they apply to API documentation. Our results show that developers initially try to form a global understanding regarding the overall purpose and main features of an API, but then adopt either a concepts-oriented or a code-oriented learning strategy that API documentation both needs to address. Our results also show that general quality criteria such as completeness and clarity are relevant to API documentation as well. Developing and maintaining API documentation therefore need to involve the expertise of communication professionals.

Keywords

application programming interface documentation, audience analysis, information design, technical documentation, usability

[1]Merseburg University of Applied Sciences, Germany

Corresponding Author:
Michael Meng, Fachbereich Wirtschaftswissenschaften und Informationswissenschaften, Hochschule Merseburg, Eberhard-Leibnitz-Straße 2, 06217 Merseburg, Germany.
Email: michael.meng@hs-merseburg.de

Introduction

Application programming interfaces (APIs) expose services or data provided by a software application through a set of predefined resources, such as methods, objects, or URIs (Stylos, Faulring, Yang, & Myers, 2009). By using these resources, other applications can access the data or services without having to implement the underlying objects and procedures. APIs are central to many modern software architectures, as they provide high-level abstractions that facilitate programming tasks, support the design of distributed and modular software applications and the reuse of code (Robillard, 2009).

As Myers and Stylos (2016) point out, all modern software makes heavy use of APIs. Instead of programming functionality from scratch, the fundamental task of software developers now often is to "stitch together" functionality that existing APIs provide (Stylos, 2009, p. 4). Therefore, learning to use new APIs is an everyday task many developers face. To assist in this task, APIs are typically shipped with documentation, such as programmer's guides, cookbooks, tutorials, and API reference documentation (Mihaly, 2011; Watson, 2015). This article is concerned with the question which information API documentation should contain from the perspective of software developers to support their initial learning efforts effectively.

The question of API learnability is closely linked to the more general issue of API usability (Clarke, 2004; McLellan, Roesler, Tempest, & Spinuzzi, 1998). Numerous studies have demonstrated that some APIs are more difficult to learn and to use than others (Myers & Stylos, 2016). While some of this difficulty is likely due to properties inherent to the API, such as the domain it covers, specific aspects of API design contribute to the difficulty as well. By now, many studies have identified and investigated relevant design aspects and demonstrated that APIs can be learned and put to use more easily if the API design is tuned to the way developers work and matches the expectations they form toward an API (McLellan et al., 1998; Stylos & Clarke, 2007; Zibran, 2008).

Besides aspects of API design, the growing interest in API usability has also generated growing interest in API documentation which reflects the fact that documentation has critical impact on usability by enabling users to solve tasks effectively and efficiently (Alexander, 2013; Guillemette, 1989; Redish, 2010). The findings from the literature suggest that API documentation plays an important role in learning and using APIs (Dagenais & Robillard, 2010; Lethbridge, Singer, & Forward, 2003), but often does not seem to fit the information needs and expectations of API users. Robillard (2009) as well as Robillard and DeLine (2011) conclude from a series of interview and questionnaire studies conducted among software developers that the lack of adequate documentation resources constitutes the most severe obstacle when learning a new API. As their studies show, documentation-related problems outrank other potential problems developers face when approaching a new API, such

as a complex API structure that is difficult to understand, insufficient technical background, or problems with the technical environment. Complementing these findings, Steinmacher, Chaves, Conte, and Gerosa (2014) provide evidence that incomplete or incomprehensible documentation is one of the main barriers preventing open source projects to attract new contributors.

It has also been observed that developers tend to prefer information sources outside the official API documentation in case they face a problem that they need to solve. For example, Parnin (2013) reports results from an informal study that tracked the search behavior of Android developers over a period of 11 weeks. The study found that developers visited StackOverflow, one of the currently most active developer forums, three times more often than the Google site hosting the official Android documentation. Parnin's results converge with an observation emphasized by Treude, Barzilay, and Storey (2011) according to which answers on StackOverflow often become a substitute for official product documentation. Sillito and Begel (2013) also note that the developers whose learning activities their study investigated primarily relied on blogs and StackOverflow posts that were not authored by the platform owner.

Finally, Maalej and Robillard (2013) revealed that a lot of information contained in API reference documentation tends to be of little value. The authors performed a functional classification of information units in the Java SDK and .NET API references and found that the second greatest share was for information units classified as ''non-information,'' which they define as ''a section of documentation containing any complete sentence or self-contained fragment of text that provides only uninformative boilerplate text'' (Maalej & Robillard, 2013, p. 1268), especially for methods and fields.

All these findings suggest that more research is necessary to better understand which information is necessary to enable developers to get into a new API efficiently and effectively. Researching these information needs is a crucial part of audience analysis. In the remainder of this article, we report the results of two empirical studies—an interview study and a questionnaire survey—that contribute to a ''feedback-driven approach to audience analysis'' (Schriver, 1997, p. 160) by providing qualitative and quantitative evidence on which information developers expect and look for when they start to work with an API unfamiliar to them. As we discuss in more detail in the next section, our study builds upon the work by Robillard and coworkers (Robillard, 2009; Robillard & DeLine, 2011; Uddin & Robillard, 2015) on documentation-related obstacles and deficiencies with existing API documentation that hinder learning. Our work extends those studies as well as other previous studies on API documentation in taking a broader approach that discusses information needs and information-seeking behavior of developers in the context of the process of learning an API.

Learning an API involves understanding the functionality exposed by an API as well as learning the API elements and how to coordinate them in order to bring the required functionality about (Stylos & Myers, 2006).

Being an everyday activity of developers, such learning can be conceived of as "self-directed learning" in the sense of Knowles (1975). According to Knowles (1975), self-directed learning is a process in which individuals take the initiative in, among other things, formulating learning goals, identifying useful resources for learning, and adopting appropriate learning strategies (p. 18). Our study helps to understand information needs of software developers by eliciting data on learning goals developers set for themselves when starting the learning process, as well as specific activities they initiate and information resources they turn to in order to accomplish those goals.

A first issue we address in our study therefore concerns the initial questions developers raise when approaching a new API and the first steps they usually take at the beginning of the learning process. A second issue that we focus on relates to the information resources developers use in order to get the information they need. Special interest in this respect will be devoted to the use of documentation provided with an API, but other information resources such as developer forums will be taken into account as well. A third issue addressed in our study pertains to the quality criteria developers apply to API documentation. Deeper knowledge of these aspects can be used to derive evidence-based recommendation regarding the content and the presentation format of API documentation, thus supporting the planning stage in information design (Redish, 2000). It will also be of value to prioritize API documentation activities and may ultimately help to provide more usable APIs.

Overview of Related Work

A considerable body of empirical research on the role of documentation for software systems, programming frameworks, and APIs has accumulated over the past few years. The academic literature is complemented by a number of blogs on API design and API documentation, such as Tom Johnson's "I'd rather be writing" (http://idratherbewriting.com), that also share important insights and experiences. This section summarizes major findings from the literature on the questions targeted by our own study.

Which Goals and Strategies Do Developers Adopt When Using an API?

Sillito and Begel (2013) describe the learning process of 10 developers beginning to program on the Windows Phone 7 platform at home in a private setting. Using Knowles' model of self-directed learning as a reference, they analyzed learning goals and learning strategies driving the learning process on the basis of interviews and diary entries. As Sillito and Begel note, learning goals formulated by the developers were strongly dependent on particular features of the application they intended to build. Their subjects did not attempt to learn

the API as a whole, but studied specific parts relevant to their project. Learning strategies were similarly selective and focused on getting particular API functions to work or identifying workarounds in case of problems instead of learning how to use the function in general.

The findings reported in Hou and Li (2011) suggest that developers actively using a framework also predominantly seek solutions to a specific problem and are less interested in getting an overall understanding. Analyzing 172 discussion threads containing about 800 messages on a Java Swing forum, Hou and Li (2011) found that in the majority of cases (86 of 172), developers wanted a fix for their current problem. Only 16 threads were concerned with gaining a better understanding, for example, regarding how a specific method works or why a particular program behavior occurred.

A similar conclusion was reached by Maalej, Tjarks, Roehm, and Koschke (2014) who performed both a qualitative and a large-scale quantitative study to explore and assess strategies developers follow to comprehend software programs as well as channels they prefer to access and share knowledge about software. According to Maalej et al. (2014), developers tend to avoid a deeper understanding of the code. They want to get their task done rather than comprehend the software. The authors conjecture that this finding might be due to the fact that "software comprehension is a hard and time consuming task and consequently is avoided whenever possible" (p. 28). In sum, these and other studies on this topic (e.g., Nykaza et al., 2002) collectively show that the learning goals and strategies of developers are highly selective and very much focused on resolving specific tasks at hand.

Which Information Resources Do Developers Prefer?

Robillard (2009) reports the results of a questionnaire that focused on obstacles which hinder API learning. One set of questions was intended to elicit information regarding information resources developers turn to during the initial learning process. The 83 responses revealed documentation as the most preferred information resource, mentioned by 78% of the respondents, whereas 55% use code examples, 34% indicated to prefer learning by directly experimenting with APIs, and 29% by communication with colleagues. Communication is reported to be preferred over documentation in studies focusing more broadly on software development and maintenance (Ko, DeLine, & Venolia, 2007; Maalej et al., 2014). Moreover, as already discussed earlier, there is evidence that developers prefer to search the web to identify and access information resources, instead of restricting their search to information resources provided by the API or platform owner (Parnin, 2013; Sillito & Begel, 2013).

Several studies emphasize the importance of source code as an information resource in software development (Maalej et al., 2014) and software maintenance (Garousi et al., 2015; Seaman, 2002). Maalej et al. (2014) conclude that

developers regard source code a more trusted source of information than written documentation, apparently because developers view code as a resource that—in contrast to documentation—cannot be outdated. Moreover, Kim, Bergmann, Lau, and Notkin (2004) and Maalej et al. (2014) observe that source code is often cloned and reused as starting point to solve the specific problem at hand. In a similar vein, code examples have been described as an important information resource for learning a programming language or framework from which both beginners and experienced users can benefit (McLellan et al., 1998; Nykaza et al., 2002; Shull, Lanubile, & Basili, 2000; Stylos et al., 2009)

Deficiencies of API Documentation From a Developer's Perspective

Despite the growing interest in information needs and information resources, only little information is available on individual experiences developers have made with API documentation, typical problems noted by developers and specific issues observed during use.

Maalej et al. (2014) described as problems that information relevant to solve a specific problem may be scattered across various sources, and that answers are sometimes too general and apparently not easily transferable to the specific working context of the developer. Lutters and Seaman (2007) examined the use of documentation in software maintenance environments using storytelling as a data collection technique. According to their results, structural properties of documentation (e.g., tables of contents and indices) are crucial with respect to the ability of maintainers to make use of it.

A comprehensive investigation of documentation-related issues that hinder API learning has been reported in Robillard (2009) and Robillard and DeLine (2011). Robillard (2009) describes results of a survey among Microsoft developers who asked participants to name and comment on API learning obstacles. As already mentioned earlier, the study identified several areas that make learning an API hard, such as issues with API structure and design, insufficient background on part of the developers, or problems with the technical environment. However, the majority of obstacles mentioned by the respondents refer to learning resources, such as issues with API documentation and code examples. The results of a follow-up survey (Robillard & DeLine, 2011) confirm that documentation-related obstacles are perceived as the most severe, followed by API structure, technical environment, and other issues. In addition, Robillard and DeLine (2011) discuss qualitative findings from interviews designed to explore the nature of documentation-related issues in more details. The findings reveal several specific aspects that are lacking or covered insufficiently in API documentation, such as a description of the rationale behind API design decisions, code examples that show how individual API elements pattern together, and resources that help developers match usage scenarios to API elements.

Our study uses similar research methods and, with API documentation, addresses the same domain. However, our study complements and extends the work of Robillard and coworkers by focussing more specifically on information needs articulated by developers and by discussing these information needs in the context of learning goals, learning strategies, and information resources used for learning. Moreover, we use the survey not only to assess broad categories, such as high-level and low-level API docs, but also to address the relative importance of specific factors that our interviews identify as potentially relevant.

Our Study

Goals

We followed a two-stage approach that combined both qualitative and quantitative methods, similar to the approach taken in Robillard (2009) and Robillard and DeLine (2011). The main goal of our study was not to test specific hypotheses about learning strategies, information-seeking behavior, or document usage. Instead, the overall goal was to collect experiences and opinions from developers in order to get a better understanding of their learning strategies, the types of information they look for during learning, the resources used to foster learning, and potential obstacles that hinder learning.

Toward this goal, we first ran a series of expert interviews with junior and senior developers to collect feedback arising from their work with APIs and API documentation. This feedback was used to identify issues that are relevant from a developer's point of view and grounded in their practical experience. In other words, the idea of the initial phase of our study was to let the developers talk and then see which issues they bring up.

However, on the basis of the data generated by the interview, it is not possible to draw conclusions regarding which issues point to general problems and which are rather individual. We therefore designed a questionnaire as a follow-up to the interviews. The main goal of the questionnaire was to test the main findings from the interviews against a larger population in order to better understand their relative importance and relevance.

Procedure

Interview. Interviews are a data gathering technique that is very popular in qualitative research. Interviews are inquisitive and involve a direct interaction between researcher (interviewer) and participant. The form an interview can take and the process of conducting them vary greatly depending on the research goals pursued (Bogner, Littig, & Menz, 2009; Lethbridge, Sim, & Singer, 2005).

The form of interview chosen for our study was the semistructured interview. Semistructured interviews combine features of structured interviews and

unstructured interviews (Hove & Anda, 2005). They are more formalized than unstructured interviews in that they employ a catalog of questions which are used to influence the course of the interview and to make sure that aspects which are already known to be relevant in the context of the research question are systematically covered. However, unlike structured interviews, they mainly contain open-ended questions and thus leave sufficient freedom to uncover issues that have not been anticipated by the researcher. In a semistructured interview, the questions are designed to encourage the participant to elaborate on a subject matter of interest. Moreover, compared with structured interviews, the course of the interview is more flexible, and the interviewer can deviate from the script (Lethbridge et al., 2005).

Our interviews were conducted using a catalog of 45 questions. According to their function, these questions fell into three groups: background questions, starter questions, and thematic questions. Background questions were asked at the beginning of the interview. They were designed to collect or verify demographic data regarding age, current role, technologies used recently, professional background, and level of experience. After this initial phase, one or more starter questions were asked to get the conversation going and create a friendly, open atmosphere. The starter questions also led to a first impression regarding how expressive the respective participant was and how much effort would be necessary to ensure a smooth conversation.

Thematic questions covered six areas of interest. For each area, a set of two to five trigger questions was defined. Trigger questions were intended to introduce a topic of interest and to encourage the participant to elaborate on that topic. For example, to trigger a conversation about learning strategies when getting into a new API (one of the thematic areas), we started with the question ''You have to get into a new API. What are your first steps?'' The question ''What do you expect from good API docs?'' was a trigger question for the thematic area ''API quality criteria.'' In addition, we prepared a set of follow-up questions designed to dive into a topic more deeply in case the participants were less responsive. Care was taken to ensure the questions were worded clearly. The interview procedure and the question catalog were pretested in two sessions and refined based on the lessons learned.

We interviewed 17 developers: 15 participants were recruited from two partner companies, and 2 participants were recruited from the information technology department of our university. Participants were not paid for participation. Statement of consent was obtained from each participant before the interview. To make sure that our interviews generate a broad set of findings, we used three different categories to drive the selection process. First, we asked for participants that are practically involved with programming tasks, but nevertheless serve different roles, including software engineer, consultant, or software tester. Second, we made sure to include participants with varying level of professional experience. Participants therefore included both developers at junior and senior

level, with professional experience as developer ranging from less than a year to 20 years (mean: 10 years, median: 8 years). Finally, selecting participants from different companies and institutions ensured that our participants varied regarding the technologies, frameworks, and programming languages they were currently involved with (Java and frameworks from the Java eco system, .Net, C#, PHP, JavaScript).

The interviews were conducted face-to-face at meeting rooms provided by the respective partner companies or the university. Following a recommendation of Hove and Anda (2005), each interview was conducted by two interviewers with different roles. One interviewer opened the interview and took the lead during the interview. The second interviewer observed the interview, took notes, monitored the question catalog in order to make sure that the interview followed the predefined path, and occasionally joined the conversation in case she felt that the answers provided by the participants were not sufficiently clear. All interviews were held in German, the native language of the participants. The interviews lasted about 1 hour each.

Questionnaire. Questionnaires are primarily used to collect data for quantitative analyses. As mentioned earlier, the main purpose of the questionnaire was to follow-up on issues that emerged as relevant from the interview in order to better understand the relative importance and generalizability of the these issues. Note that the questionnaire was not designed to enable a rigid statistical analysis to test specific hypotheses. Instead, the primary aim was to observe patterns and trends.

The questionnaire was administered online. A software package (EvaSys, http://www.evasys.de) hosted on a university-owned server was used to generate the questionnaire and to collect the data. The link to the server was distributed to contacts at several software companies in our partner network and forwarded by those contacts to development teams that represent the target audience of API users.

The questionnaire consisted of 39 questions, including 7 questions eliciting demographic data on age, first language, job role, programming experience, and experience with APIs, as well as 32 thematic questions. Of the thematic questions, 18 questions corresponded to topics discussed in the interview, such as the initial questions raised when starting to work with an API or information sources used or quality criteria applied to API documentation. The remaining 14 questions were exploratory in nature and intended to potentially reveal additional areas that future research might take up. Completing the entire questionnaire took approximately 20 minutes.

The questionnaire was returned by 114 participants. Data from two participants were excluded from the analysis: One participant stated to have no experience with any of the API types that we offered as answer (see Table 1), including the option ''Other APIs.'' The data for another participant included

Table 1. Means and Medians for Experience With Different API Types ($n = 112$) Rated on a Scale From 1 (=*No Experience*) to 5 (=*Much Experience*).

	Class-based APIs (e.g., Java)	Library-based APIs (e.g., Javascript)	Web APIs (e.g., REST)	Other APIs (e.g., hardware APIs)
Mean	4.45	3.46	3.65	2.23
Median	5	4	4	2

API = application programming interface.

uninterpretable data, raising doubts regarding data quality. Hence, data from 112 participants were entered into the analysis. Of the 112 participants, 104 were male and 8 female. The average professional experience was 12 years (median: 10 years), with experience ranging from 1 year to 45 years; 92 of the respondents classified themselves as developers, 38 as team leads, 22 as software testers. Thirteen respondents included additional roles such as system architect or technical consultant. The first language was German for 110 respondents, with the other two providing Dutch and Greek as answer, but stating that they were fluent in German. To better understand the technical background of the respondents, we included the question "Which programming languages have you been working with recently?" with multiple answers possible. The programming language mentioned most often was Java (68), followed by JavaScript (53) and C# (41).

To assess the amount of experience our respondents have with APIs, we asked participants to rate their experience with different API types on a scale of 1 (=*no experience*) to 5 (=*much experience*; see Table 1).

The data show that experience with modern API types is moderate to high. Note that the data regarding API experience correlate well with the data regarding the programming languages used. We conclude that respondents have sufficient experience to answer questions about API usage.

Data Analysis

Interview. All interviews were audio recorded, resulting in about 18 hours of recording material. The audio records were transcribed using standard text processing software. The transcripts were verbatim but omitted all paraverbal and nonlinguistic content, such as fillers (*ummhs*, *ähmms*), pauses, special emphasis, slips of the tongue, and other speech errors.

For analysis, our study followed basic principles of qualitative content analysis, a technique to analyze and interpret qualitative data developed by Philipp Mayring (Mayring, 2014, 2015; see also Kuckartz, 2014). Qualitative content analysis aims at structured and rule-based processes in order to maximize the validity of results drawn from raw data such as interview transcripts. Our analysis

proceeded in two main steps. First, we reduced the transcripts individually for each participant to a set of propositions that reflect the core contents of the interviews by paraphrasing and generalizing text passages as well as merging paraphrases that refer to the same topic. We then coded the resulting paraphrases across participants, using the thematic areas that structured our interviews as main coding categories.

The analysis process described here was tested and refined on the basis of the two pilot interviews conducted in advance to the main data collection phase. The main interviews were executed over a period of 2 months. Analysis started after all interviews had been completed. Content analysis was carried out using standard spreadsheet software, summarizing the original units from the transcript and the corresponding paraphrases as well as the results from paraphrasing, generalization, and reduction by participant in a set of related tables. This ensures that it is possible to trace the verbatim passages in the transcript that underly a specific generalized statement.

Questionnaire. Data collection ran over a period of 2 months as well. Responses were stored automatically by EvaSys and then exported in ASCII format for further analysis. All descriptive analyses reported in this article were conducted using the R software package (version 3.3.2; R Core Team, 2016).

Results

Which questions come first when getting into a new API?. Responses regarding the questions that developers ask themselves first when getting into a new API revealed a variety of both conceptual and practical issues. The answers show that developers initially approach new APIs with two different goals in mind. One goal is to arrive at a decision whether the API should be selected in the context of a specific task. A second goal is to actually use the API in order to perform the tasks. These two goals do not necessarily fall together. For example, several developers mentioned that they are typically requested to perform a task with an API that has already been selected.

Regardless which of these two general goals are set, initial questions concern the functional features offered by the API. Developers look out for information that provides a quick overview of what the API is about and the tasks it can be used for.

> Initially, I check what the API can do. It is important to know whether what the API can do fits to what I actually intend to do. (Ich guck erstmal, was kann die API leisten. Wichtig ist es zu wissen, ob es das trifft, was ich damit machen will, P13).

> Initially, I'm interested in getting a general overview. I need a high-level understanding of the goals and the tasks of the framework. (Also für mich ist ganz am

> Anfang erst einmal interessant, einen groben Überblick zu bekommen, also ein grobes Bild darüber, was die Ziele und was die Aufgaben von diesem Framework sind, P07).

A similar point was raised by P03. In addition to getting an API feature overview, this developer also indicated interest in information about limits of the API with respect to a certain task domain, and regarding prerequisites for using the API.

> The first thing, of course, is the feature set of the API. I would definitely check for a central document that provides a general overview of what I can do with the API, but also what I cannot do with the API and which external prerequisites possibly hold. (Also das erste ist natürlich klar der Funktionsumfang, ich würde auf jeden Fall schauen, ob es ein zentrales Dokument gibt, wo ein allgemeiner Overview gegeben wird über die ganze Sache, was kann ich mit der API machen, was kann ich auch nicht machen, was gibt es vielleicht für Voraussetzungen, externe Voraussetzungen, um die API zu benutzen, P05).

As part of getting a broad overview of the API, developers look out for information regarding the platform on which the API is running, such as .NET or Java, possible interactions with other platforms, API licensing conditions, the programming languages that can be used to interact with the API, and the overall programming workflow, that is, how to integrate the API, how to send requests to the API and how to process the response.

> First question probably is about the platform, i.e. on which platform does the framework or the API run. Second are the dependencies to other platforms. These two points are most important. And then, perhaps, the language on which the API is built. (Als erstes vermutlich die Plattform. Also wo läuft dieses Framework oder diese API. Als zweites welche Abhängigkeiten gibt es eventuell zu anderen Plattformen. Also das wären die zwei wichtigsten und dann vielleicht in welcher Sprache das Ganze umgesetzt ist, P14).

Another initial question at conceptual level concerns the architecture of the API, its components, and the general request-response behavior of the API. Again, a brief overview is looked for to find this information.

> Well, I wonder how the component is structured. Ideally, there is a section in the documentation describing how the API works in general. Hence, an introduction, a brief overview. (Also ich überlege natürlich, wie die Komponente aufgebaut ist. Im Idealfall sollte da in der Dokumentation ein Bereich sein, der beschreibt, wie das in groben Zügen funktioniert. Also ein Einführungsteil, Überblick, P17).

Note that on this aspect, our interviews also revealed exactly the opposite statement. For example, when discussing the relevance of an API architecture overview, one of the developers replied:

> No, I'm not interested in this. I want to see success immediately, i.e. not just reading something, but trying things out right away. (Hat mich bis jetzt eher weniger interessiert, wenn, dann möchte ich schon Erfolge sehen, also nicht nur irgendetwas durchlesen, sondern gleich etwas ausprobieren, P16).

Beyond conceptual questions, various additional aspects were mentioned that are rather practical in nature and that indicate a desire to get productive with the new API as soon as possible. For example, one of the developers explained that he was looking for resources that would enable him to get the API quickly up and running.

> The first question of course is which resources are available, such as source code or documentation. And then, perhaps, what needs to be installed, how to install it and how to try and test the API. (Ja die erste Frage ist natürlich was gibt's da an Ressourcen, verfügbaren Ressourcen dazu. Quellcode, Doku was auch immer. Ja, dann vielleicht wie und wo ich das installieren kann, ausprobieren kann, testen kann, P03).

In a similar vein, other developers mentioned that they would specifically search for examples showing how to implement their special use cases, instead of looking for high-level information on how to program with the API.

> How do I use the API? More specifically: What do I need to do in order to get the things done which I want to accomplish? (Wie benutze ich es? Wie erfülle ich, was muss ich tun, um das, was ich damit erreichen will, zu tun?, P15).

As pointed out by P15 in this respect as well, it is of interest to quickly get access to information regarding the method calls, parameters, and objects involved in solving a particular use case, and how these objects are processed.

As the discussion so far has illustrated, the first questions raised by developers when approaching a new API are partly conceptual in nature and partly very practical and driven by the specific goals the developers want to achieve using the API. Important themes emerging from the interview were included in the questionnaire as fixed answer options to the questions "Suppose you have to get into a new API: Which question comes up first?" and "Which question comes up second?" The results regarding the question that comes up first are shown in Figure 1.

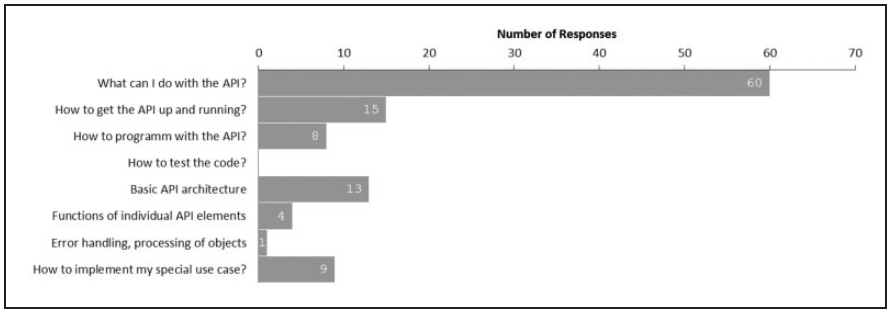

Figure 1. Frequency of answers selected in response to the question "Suppose you have to get into a new API: Which question comes up first?" One answer per participant ($n = 110$).

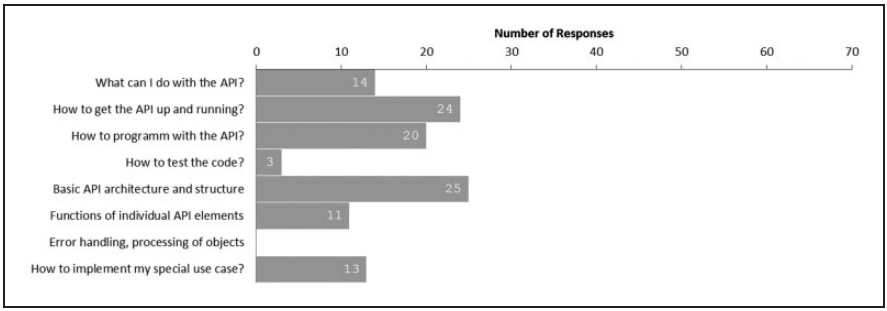

Figure 2. Frequency of answers selected in response to the question "Suppose you have to get into a new API: Which question comes up second?" One answer per participant ($n = 110$).

As Figure 1 shows, the answer option "What can I do with the API?" clearly emerged as the winner, suggesting that conceptual issues such as the overall purpose of the API and the functional features offered by the API dominate initially. These questions arise both when evaluating the API against the goals to be accomplished and also when starting to learn the API after the decision to use it has been made.

The pattern of questions that follow on the initial question regarding the overall purpose and the API feature set is more diverse, as shown in Figure 2.

Once the purpose of the API has been clarified, conceptual questions, for example, about the basic API architecture and the programming workflow are at similar level with more practical and task-driven questions such as "How to get the API up and running?" or "How to implement my special use case?" We conclude that learning goals at conceptual level dominate initially.

However, once an initial understanding regarding the purpose of the API and its feature set has been formed, two different pathways seem to emerge: Developers either want get going quickly or they want to build a more thorough understanding of the API before starting to do real work with the API.

How do you proceed when learning a new API?. The second thematic area of the interview attempted to reveal information regarding the learning strategies adopted by developers. Trigger questions of this thematic area were aimed at specific activities which developers initially undertake. Do they try to work with code right away? Or do they take a more systematic approach and try to develop a more thorough understanding of the API before starting programming? Consistent with the findings described in the previous section, answers from our interview partners provided evidence for both these strategies.

Some developers seem to follow a top-down approach that is concepts oriented. These developers spend some effort to build a deeper and more thorough understanding of the API before they attempt to implement particular use cases.

> I first need to understand the bigger issues and tie them together. Details follow later. (Erst einmal die groben Sachen, man bringt das zusammen und dann geht man ins Detail, P14).

Developers in this group pointed out that they systematically search and regularly consult documentation provided by the API supplier. For example, they look for documents like a technical architecture overview explaining general concepts. They then start by working through a "Getting Started" project or a beginner's tutorial, or by building a simple prototype themselves.

> When I get into an API, I need documentation that explains the concepts. But along with these concepts, I need examples that illustrate how the concepts are put to use. Simple scenarios as examples. (Wenn ich mich einarbeiten würde, bräuchte ich natürlich eine Dokumentation wo Konzepte erklärt werden, und zu diesen Konzepten aber auch Programmierbeispiele, also gewisse Szenarien der Anwendung. Als Beispiele einfache Szenarien, P07).

Not surprisingly, the efforts required by this approach vary depending on complexity of the API to be learned.

> For more complex topics, the learning effort increases. A basic understanding: How does it work? What is it anyway, what is it doing, and why? This leads to greater efforts initially when trying to get into the system. (Bei komplexeren Dingen ist der Lernaufwand erst einmal größer. ... So ein Grundverständnis: Wie tickt das? Was ist das überhaupt? Was tut das? Warum? Da ist dann die entsprechende Lern- und Einarbeitungsphase natürlich entsprechend größer, P08).

On the other hand, there are developers that take a bottom-up approach that is code oriented. One of the developers put it simply like this:

> I'm writing code and therefore I'm looking for code. (Ich schreibe Code, da suche ich nach Code, P15).

Typically, developers from this group want to start coding right away. Participants mentioned that they start learning by trying examples which they extend step-by-step. They check documentation only if they cannot find a solution on their own.

> I take an approach which is more intuitive. For example, I get an example and try to rebuild it. And if things go wrong at some point, then I look for documentation. Things should be self-explaining. Only if things do not work as expected, I resort to documentation. (Das ist eher ein intuitives Gefühl wie ich an so eine Sache rangehe. Ich hole mir, sagen wir mal, ein Beispiel und versuche das nachzustellen. Und dann, wenn es irgendwo hapert, dann gucke ich: Wo gibt es die Doku? Also eigentlich sollte es, falls möglich, selbsterklärend sein und wenn es dann irgendwo hakt, dann greife ich erst auf die Doku zurück, P11).

Developers using a code-oriented approach are strongly task driven in their learning activities, as one developer pointed out directly. This strong focus on tasks may lead to a rather unsystematic hunt for information somehow related to the problem which needs to be solved.

> Well, learning an API in general has never been a task of mine. There has always been a specific task, such as: do this and use that API. And then you somehow try to deduce from the API how to do just that, or you search in the documentation or in discussion forums to find out how it could be done. (Also, so generell einarbeiten hatte ich bis jetzt noch nie. Es gab immer eine Aufgabe dazu, also: Mach das und das und nimm das. Und dann versucht man halt, aus der API herauszulesen bzw. in der Dokumentation nachzuschauen oder auch in Foren, wie man die Aufgabe irgendwie erledigen kann, P16).

Developers using a code-oriented approach will look out for an example that can potentially serve as a basis to solve their problem, and they will stop learning about the API as soon as their problem is solved, regardless of whether the API is simple or complex.

> The docking library of our client can do a lot, but we only wanted to use the "Move Window" feature. Therefore, we took the "Move Window" example and built our code starting from that example. It was working, that was it. (Unsere Docking-Bibliothek im Client kann viel, ... aber wir nutzen halt nur diese

> "Fenster-Rumschiebe"-Funktion. Da haben wir halt das "Fenster-Rumschiebe"-Beispiel genommen und haben es eingebaut anhand des Beispiels. Es läuft, fertig, P15).

Comments included with code examples play an important role for the code-oriented approach, as can be inferred from the following statement:

> I took an example and extended it step by step using the hints provided along with the respective functions, until my problem was mainly covered. (Hier habe ich das Beispiel genommen und dann langsam Schritt für Schritt versucht zu erweitern mit den Hinweisen, die bei den jeweiligen Funktionen standen, bis das Problem, das ich hatte, grob umschlossen war, P12).

To follow up on the topic of learning strategies and learning activities, the questionnaire included the question "You have to get into a new API. What are the first steps you take?" This was a closed question that offered seven answer options based on statements derived from the interviews. Participants were allowed to select up to three answer options. The frequency distribution across the answer options is shown in Figure 3.

As the results of the questionnaire reveal, two activities are head-to-head which represent the two learning strategies that became apparent from the interviews. The option selected most often ("I work through code examples") is an activity indicating a bottom-up, code-oriented approach. On the other hand, the answer option "I read a 'Getting Started' document" was selected almost as often. This answer option describes a learning activity that is indicative of a more systematic, concepts-oriented top-down approach.

Typical problems when learning a new API. Inspired by the work reported in Robillard (2009) and Robillard and DeLine (2011), we included questions in

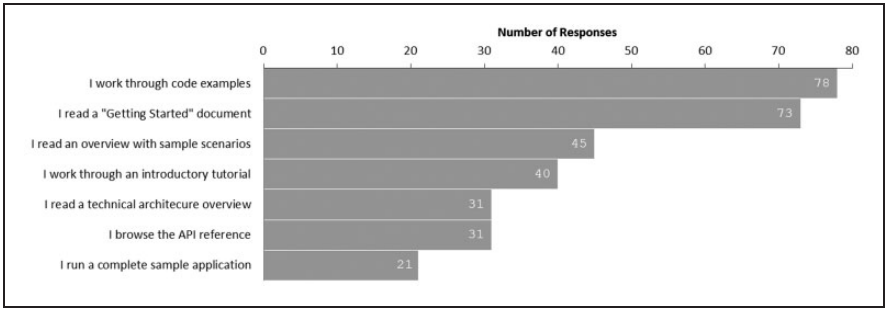

Figure 3. Frequency of answers selected in response to the question "You have to get into a new API. What are the first steps you take?" Multiple answers (up to three) possible per participant.

our interview to elicit information regarding typical problems or obstacles that developers face when they have to get into a new API. Some interview partners mentioned that there are always problems if the task is to learn a new API, but that the nature of these problems varies from case to case (e.g., P15). However, it quickly became apparent that there are certain areas that pose learning challenges rather regularly. These areas are in part conceptual in nature, such as the problem to understand the intended API usage, but very often point to practical issues like inaccurate documentation or code examples that do not work.

At conceptual level, an issue that recurred in our interviews relates to the way the API should actually be used. Many developers expressed that they find it sometimes difficult to identify the right entry point to get started with the API. They have a problem they need to solve, they know the API will potentially solve their problem, but they do not know where and how to begin.

> In most cases, the problem simply is to find the entry point. To get something running in the first place is always the big issue. Once you made the first call to the API, everything else is not that difficult. (Meistens ist das Problem, den Einstieg zu finden. Das ist immer das ganz große Thema, da überhaupt erst mal was ans Laufen zu kriegen. Wenn man dann erst mal den ersten Aufruf gemacht hat, dann ist das weitere nicht mehr so schwierig, P03).

> Where to start, and how to begin in this network of types, classes methods, and the like? (Wo geht es los, wo fange ich an in diesem ganzen Netz von Typen, Klassen, Methoden usw?, P11).

> Well, in my view the issue is to somehow make the first dance moves, whatever exactly it is that needs to be done. To find a way you can go, to find the entry point that will lead you to a first experience of success. (Naja, das ist in meinen Augen immer erstmal die Sache um die ersten einfachen Tanzschritten mit dem Ding zu machen, was auch immer man da tun muss. Also dort überhaupt erstmal einen gangbaren Weg zu finden, den Einstieg zu finden und erstmal ein Erfolgserlebnis zu haben, P09).

A somewhat different set of problems at conceptual level may arise if the API is built on domain-specific concepts that require expert knowledge outside programming itself. For example, one developer described a genetics project in which her task was to use an API for DNA sequencing. Understanding the workflow and finding out which classes and methods to use for her specific purpose posed a challenge simply because she did not have any background in genetics, including the domain-specific vocabulary that was used to structure API resources.

> Here is an example, a private project: an API that somehow processes DNA sequences. I didn't understand it, simply because I didn't know the domain.

> I first had to get into all the specific vocabulary, how those amino acids are called. There were certain terms which I simply didn't know. (Ich habe ein Beispiel, das war eine Privatentwicklung. Eine API die DNA-Sequenzen irgendwie verarbeitet. Ich habe es nicht verstanden, weil ich einfach die Domäne nicht kenne. Ich musste mich erst einmal in das ganze Vokabular einarbeiten. Was heißen diese Aminosäuren und so weiter. Da waren bestimmte Bezeichnungen, die ich einfach nicht kannte, P14).

Related to the question regarding the proper entry point to get started, various problems were reported with integrating the API in a specific development environment, understanding technical requirements for using the API, such as licensing conditions, and ultimately being able to send requests to the API and to receive and process responses.

> My initial challenge was to get something to work. To configure the development environment can become an important issue, especially if you don't know what to do and where to click. (Ich hatte am Anfang erst einmal Schwierigkeiten, überhaupt was auf die Reihe zu bekommen. Dass man die Umgebung einrichtet, das ist ja manchmal schon ein größeres Problem, wenn man nicht weiß, wo man hinklicken muss, P16).

> For APIs, the question often is how to integrate them. And if they require a license: how do the licenses work? And if I have got a license: how do I make sure that everything works properly? And are there code examples illustrating typical API usage? (Bei APIs ist oft die Frage, wie man sie überhaupt einbindet. Wenn sie lizenzpflichtig sind, wie funktionieren überhaupt die Lizenzen? Und wenn ich die Lizenzen habe: Wie sorge ich dafür, dass alles funktioniert? Und gibt es ein Code-Beispiel, was zeigt, wie die typische Verwendung ist, P17).

As the last quote already suggests, specific documentation resources that are designed to support the onboarding process, such as "Getting Started" documents or sample projects, are routinely referred to in order to solve practical issues regarding API integration. However, responses in the interviews revealed that those resources are sometimes unreliable, which the easily leads to frustration on part of developers.

> Sometimes, even the Getting Started project or the example does not work and then you first need to figure out why. (Manchmal funktioniert schon einfach dieses Getting Started-Projekt oder das Beispiel nicht und dann muss man erstmal schauen, warum das wieder nicht geht, P16).

> Yesterday, I tried to set up the whole system on my machine. There is one page of documentation describing how to do it. However, since the system changes

> frequently and the documentation is not up to date ... well, I have now tried for eight or nine hours, and it is still not working. (Ich habe gestern versucht, das ganze System auf dem Rechner einzustellen. Es gibt eine Seite Dokumentation, wie man das macht. Da sich das aber immer alles laufend ändert und die Dokumentation nicht up to date ist ... ich weiß nicht, ich bin jetzt die achte oder neunte Stunde dabei das einzurichten und das ist immer noch nicht fertig, P12).
>
> It is simply annoying if there are code examples available which do not run and which I can't reproduce. (Das ist dann schon ärgerlich, wenn dann irgendwelche Codebeispiele vorhanden sind, die für sich nicht lauffähig sind und die ich nicht nachvollziehen kann, P06).

As follow-up, our questionnaire included the question "What are typical problems when getting into a new API?" From the interview data, we derived 12 answer options reflecting the issues discussed earlier as well as other conceptual and practical problems mentioned in the interviews. Participants were allowed to select multiple responses, with no specific upper limit defined. In addition, a free-form field was provided which respondents could use to indicate additional problems. Results are shown in Figure 4. None of the respondents used the free-form field, which suggests that our answer options covered a significant range of problems that developers actually encounter when getting into a new API.

In line with Robillard (2009), results from our questionnaire confirm that documentation-related issues, such as documentation errors, incomplete documentation, or documentation that is incomprehensible, rank among the most important issues from the developers' perspective. They outrank both problems that are more conceptual in nature, such as problems regarding proper API

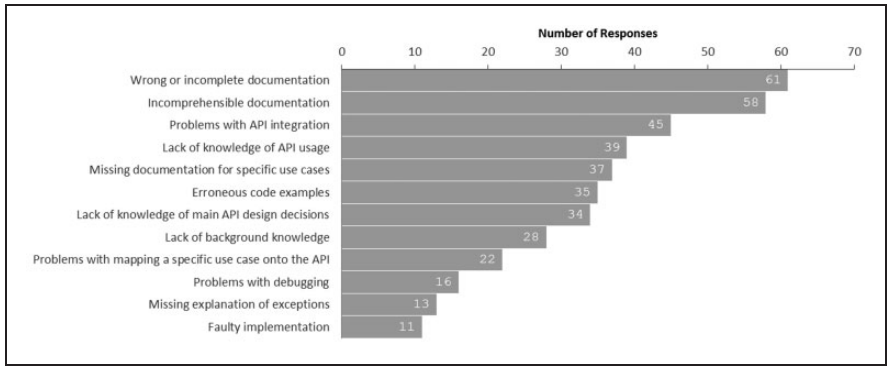

Figure 4. Frequency of answers selected in response to the question "What are typical problems when getting into a new API?" Multiple answers possible per participant.

usage or missing background knowledge in a certain expert domain, as well as practical problems, for example, problems with respect to integrate the API and get it up and running in a certain development environment.

Which information resources do you use?. As the quotes from the interviews in the previous sections already revealed, developers rely on various types of documentation resources, such as documents providing an overview of the purpose of the API and main API features, a "Getting Started" document that helps to identify appropriate entry points and to get a sense of achievement, or code examples. In our interview, we included trigger questions that specifically addressed the question which information resources developers use when learning a new API. In addition, we wanted to find out whether and to which extent the choice of information resources depends on factors such as the nature of a specific task or the level of experience.

Beyond documents already mentioned in the previous sections, such as overview documents and "Getting Started" guides, tutorials and API references were mentioned to be important information resources. Tutorials were described to be especially useful during the initial onboarding stage, when starting to deal with a new API. API references get more important at a later stage and are used to search for specific information when working on a certain task.

> If the decision has been made to use a certain API, I would be interested in a tutorial that demonstrates the basic setup and API usage in standard situations. Later, when I start to work on my specific problem, I would like to know where to find an API reference to quickly look up specific information. (Wenn fest steht, dass die API verwendet werden soll, dann würde mich ein Tutorial interessieren, wo Standard Use Case durchexerziert werden und man mal guckt, was so das grundlegende Setup ist und wie man die API in Standardsituationen verwendet. Dann natürlich, wo es eine Referenz gibt, wenn ich mich dann langsam an das Problem mache, was ich da beackern will. Wo ich dann für ganz konkrete Probleme auch gezielt und schnell Sachen finde, P13).

However, whether tutorials are used or not may also be dependent on the overall learning strategy adopted by a developer. Unless tutorials address exactly the problem that needs to be solved, they are more likely to be used by developers that follow a concepts-oriented strategy and try to develop a more thorough understanding of the API. On the other hand, tutorials will likely be ignored if a code-oriented strategy is used, and the aim is to solve a problem as quickly and efficiently as possible, as made clear by a developer describing her work with a new API to load and convert graphics.

> I have never worked with a tutorial, in the sense of you do this and that and then you know how things work. I might give tutorials to novice beginners, but this is

> more like developing an understanding of things and collecting experiences. I don't want to collect experiences with the new API. I want to know how to load graphics and convert them to a different format. (Ich habe noch nie mit einem Tutorial gearbeitet. Im Sinne von, mach mal dieses und hinterher weißt du dann wie es gedacht ist. Neuen Leuten würde ich ein Tutorial geben, aber das ist mehr so dieses Verstehen, so erste Erfahrungen sammeln. Ich will nicht erst Erfahrungen mit einer neuen API sammeln. Ich will wissen, wie ich die Grafik lade und in einem anderen Format speichere, P15).

Cookbooks containing sets of smaller recipes with a fixed problem-solution structure were mentioned as information resource as well. Not surprisingly, they are regarded particularly useful if they contain exactly the problem that needs to be solved. Another information resource emerging from the answers is the source code, if available. Source code was described as useful in case of unexpected API behavior, in order to clarify error handling and in case of exceptions.

In line with previous findings from the literature, developers also indicated that they would restrict their search for information resources not to the official API documentation or the web site of the API supplier. Instead, search engines like Google appear to be the preferred starting point to identify help in case of problems.

> Basically, I always search for resources on the Web. Google is the main starting point for this. Regardless of the question: You can't beat Google if it comes to identifying appropriate help resources. If you look for developer documentation, have a specific problem and a specific question, you will need to go to Google. (Im Endeffekt schau ich, was gibt's für Ressourcen im Netz. Google ist der Ansprechpartner schlechthin. Überhaupt so bei fast jeder Frage: Google ist einfach unschlagbar, was das betrifft. Wenn man Entwicklerdoku sucht, wenn man Problemlösungen oder Fragestellungen sucht, dann ist eine Suchanfrage bei Google das A und O, P04).

This assessment was confirmed by our questionnaire. When asked "Suppose you encounter a problem while working with an API. What do you try first?"; 53 out of 111 respondents selected the answer option "Ask Google", 37 respondents selected the option "Check documentation," and 15 the option "Try to solve the problem myself" (see Figure 5).

In addition to using search engines like Google, many developers pointed out that they frequently turn to developer forums such as StackOverflow if they need help. Several arguments were put forward why forums are popular. First, the time stamp associated with questions and responses in a thread immediately conveys an impression as to how up to date the information is. A second point was that user comments—if available—help to reassure the quality of the answer provided, and they are also felt to help phrase the initial problem

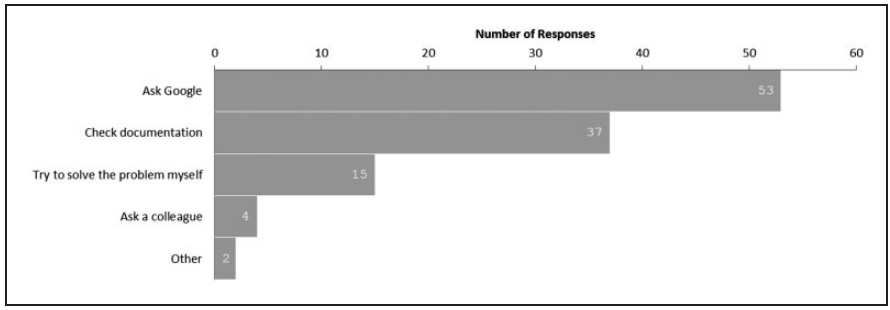

Figure 5. Frequency of answers selected in response to the question "Suppose you encounter a problem while working with an API. What do you try first?" One answer per participant ($n = 111$).

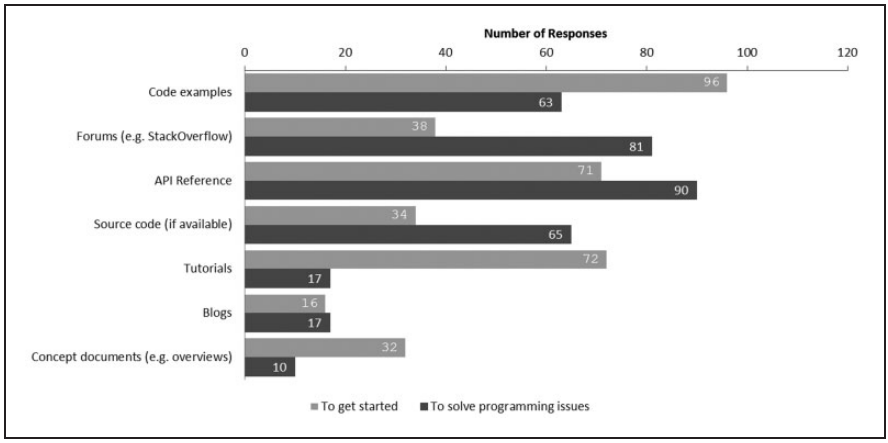

Figure 6. Frequency of answers selected in response to the question "Which information resources do you use (a) to get started or (b) to solve programming issues later on when working with an API?" Multiple answers possible per participant.

more precisely. It was also pointed out that answers in developer forums typically include useful and relevant examples. Finally, the question-answer structure of threads seems to be appealing to developers because it conveys a special sense of community working on related problems.

In the questionnaire, we followed up on these questions and asked which information resources developers use. As part of the answer, developers were asked to distinguish two usage contexts: (a) to get started or (b) to solve programming issues later on. The combined results are shown in Figure 6.

As Figure 6 shows, the use of information resources depends on the actual usage context. To get started, developers rely to a great extent on code examples and tutorials. The API reference is an important information source during the onboarding stage as well. However, it becomes even more important once developers have mastered the basics of a new API and start to work on their specific problem. Besides the API reference, developer forums get more important as well. Documents that specifically address conceptual information are of rather limited importance even in the initial learning phase.

What do you expect from good API docs? What are typical issues?. Our interview included several trigger questions that were intended to elicit details regarding the expectations that developers have toward API documentation and quality criteria they apply to API documentation.

Almost all interview partners mentioned that the most important criteria for good API documentation are that they are up to date and complete. Conversely, incomplete and outdated documentation was described as a problem that developers appear to encounter rather frequently.

> To be complete. That would be the most basic thing. (Vollständig. Das wäre eigentlich das einfachste, P11).

With respect to objects, methods, and parameters, developers expressed that they expect brief but informative explanations that go beyond merely stating what can already be derived from the method's name or signature.

> The API reference has to be complete. It should tell me what a method returns and which parameter it needs as input. For certain situations, it should also tell me: If you provide that value as parameter, then this will happen. The API reference should include everything necessary to use the API successfully, but not more. (Die API-Referenz sollte vollständig sein. Sie sollte mir sagen, was eine Methode zurückgibt und was ich als Parameter liefern muss. Sie sollte mir für bestimmte Situationen auch sagen: 'Achtung, wenn du in dem Parameter diesen Wert lieferst, dann passiert das'. Eine Referenzdokumentation sollte alles ausdrücken, was wichtig für mich ist in der Anwendung, aber nicht mehr. P10).

With respect to the content, developers expect from good API documentation that it includes code examples (a topic that will be dealt with in more detail in the next section), covers error handling in a way that supports developers in detecting the source of an error, discusses—where appropriate—alternative classes, methods, or parameters, and provides the necessary information for developers to decide which alternative to use in which situation.

Regarding organization and findability of information contained in API documentation, developers mentioned that they expect documentation to be

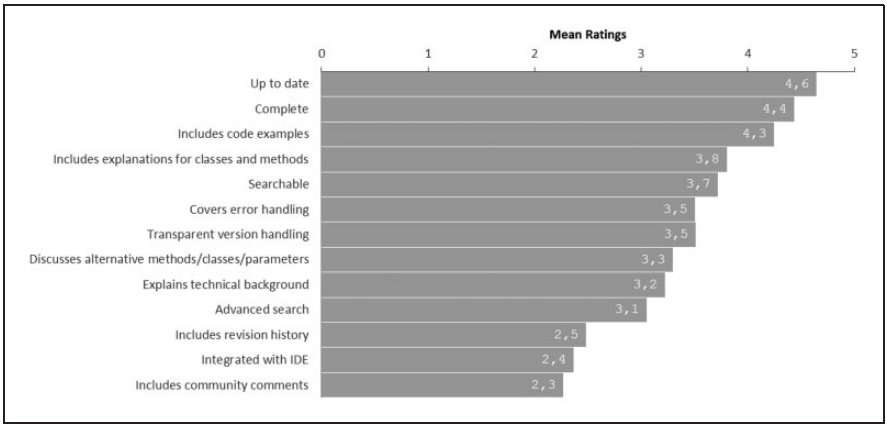

Figure 7. Mean ratings of the importance of 13 properties that were mentioned in the interviews as properties of good API documentation. Importance was rated on a scale from 1 (*totally unimportant*) to 5 (*very important*).

searchable and integrated into developer tools or the integrated development environment. Information pertaining to a specific version of the API needs to be clearly marked, and a revision history should be provided to indicate how often and when the API documentation has been updated. It was also mentioned that user comments that are integrated into the API documentation are regarded helpful and are regarded as trust signal that the API documentation has been assembled with care and is actively maintained.

To get a better understanding which of the issues mentioned in the interview are generalizable and express expectations that many developers have, the questionnaire included 13 properties of good API documentation that were derived from the interview. For each property, respondents were asked to rate the importance on a scale ranging from 1 (*totally unimportant*) to 5 (*very important*). The mean ratings for each property are shown in Figure 7.

As Figure 7 shows, two classical quality criteria for documentation emerged as the most important ones for API documentation as well: API documentation is expected to be up to date and complete. The request for code examples, for informative explanations of classes, methods, and parameters and for the possibility to search the API documentation turn out to be rather important as well. Interestingly, features such as the inclusion of community comments and—somewhat surprisingly—the integration of API documentation into the development environment are regarded less important.

In addition, the questionnaire also included the question "What are typical obstacles you face when working with API documentation?" Seven answer options were provided, that summarized statements from the interviews, such as unclear content structure and navigation options, a document structure that is

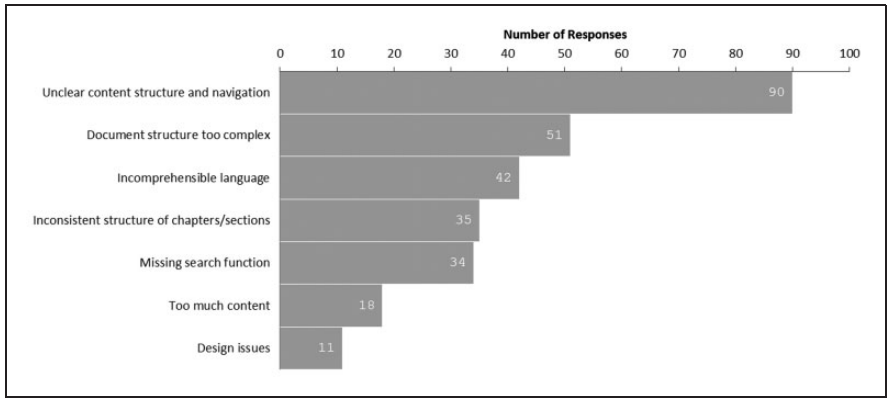

Figure 8. Frequency of answers selected in response to the question "What are typical obstacles you face when working with API documentation?" Multiple answers (up to three) possible per participant.

too complex, an inconsistent structure across chapters and sections, or incomprehensible language. Multiple responses were possible for this question and up to three answers could be selected. Figure 8 shows how often each answer option was selected.

According to Figure 8, the most salient obstacle developers face when working with an API pertains to the structural organization of the content and, related to that, insufficient cues on how to navigate the documentation efficiently. Apparently, developers often have problems finding and accessing the information they need. However, the problem is not that there is too much content, but that the content is not organized in a way that supports efficient access to information.

In addition, a free-form field was included in which respondents could enter other obstacles not offered as answer option. Problems mentioned in the free-form field included the following:

- Documentation contains errors, is not complete, or not up to date.
- Essential use cases are not covered.
- Explanations for classes, methods, or parameters are too brief and not informative.
- Search function is missing.

Role of code examples. Code examples are often mentioned in the literature as an important learning resource (Nykaza et al., 2002; Robillard & DeLine, 2011; Shull et al., 2000). The importance of code examples has also been prominent in some of the quotes discussed earlier. In addition, when asked "Do you like to

work with code examples?'' in our questionnaire, 106 of 110 respondents answered ''yes'' to that question, only 4 answered ''no.''

Several statements in the interviews relate the popularity of code examples as a learning resource to the fact that developers seem to trust code more than text, mainly because the documentation could possibly be outdated or wrong, in line with observations reported in Maalej et al. (2014).

> I trust the code more and see what it is doing. Because the comments provided with the code could be wrong because they have not been updated. (Ich vertraue eher auf den Code und guck mir an, was der macht. Denn was dazu da steht, kann auch etwas anderes sein, weil die Dokumentation nicht mitgepflegt wurde, P13).

> Code really is the best documentation. Documentation—but not the code—can be outdated or misleading. Hence, code is the best documentation, but can be more difficult to comprehend. (Eigentlich ist die beste Dokumentation der Code. Dokumentation kann veraltet sein, kann missverständlich sein, und Code ist das eigentlich nicht. Code ist die beste Dokumentation, die aber natürlich nicht am leichtesten zu verstehen ist, P01).

The interview statements pointed to several functions that code examples can serve. A function regarded as important is that they help to find entry points into an API faster.

> I have a goal I want to accomplish with the API and I have an appropriate tool. Now I need to know how I can accomplish my goal. I prefer to learn that via a code example. (Ich habe Ziel, was ich durch API erreichen will, ich habe das Tool, mit dem ich es erreichen will und jetzt muss ich wissen, wie ich es damit erreiche. Das erfahre ich am liebsten anhand eines Code-Beispiels, P15).

As this and other developers pointed out, code examples are regarded to be more informative and can be grasped faster than text.

> In most cases, you can get more out of a code example than a text. As a developer, you get into the code example faster. And you can grasp it at once more easily than a long block of text. (Und meistens kann man mit einem Beispiel eher etwas anfangen als mit dem Text. Da kommt man als Entwickler einfach schneller rein. Das kann man so auf den ersten Blick auch besser erfassen, als diesen ellenlangen Text, P14).

As another advantage, it was mentioned that code examples automatically convey information about prerequisites and dependencies.

> Perhaps something needs to be loaded first, how would you do that. Or if a function is not used independently, but in a typical context, which is not obious.

> (Vielleicht muss erst einmal etwas geladen werden, wie macht man das dann. Oder wenn eine Funktion nicht für sich alleine verwendet wird, sondern in einem typischen Zusammenhang, der sich aber nicht von selbst erschließt, P17).

Confirming findings by Kim et al. (2004) and Maalej et al. (2014), code examples were also described to often serve as starting point to produce new code for the task at hand. Code examples are copied and modified to fit the new context.

> Of course, the code snippets help to adjust the problem to your own needs and to assemble your own code from the code examples. (Die helfen schon, diese Code-Schnipsel, um eben die Fragestellung auf die eigenen Bedürfnisse anzupassen und aus den Beispielen dann schon eigenen Code zusammenzustellen, P14).

Importantly, code examples also seem to serve a signaling function. They help developers identify relevant sections in the documentation. For example, when scanning a page, developers first check the code in order to verify that the page content actually relates to their current problem.

> On a page I always check first: Where is the code? (Auf einer Dokuseite schaue ich immer: wo ist der Code?, P15).

Finally, it was pointed out that code examples help developers to get an intuitive feel for how code using an API should look like.

To be able to assess how widespread the different purposes for using code examples are among developers, our questionnaire included the question "For which purposes do you use code examples in your daily work?" Respondents had the options to choose up to three answers from a set of five answer options. Figure 9 shows how often each of the answer options was selected.

To find a meaningful entry point when starting to work with a new API and to better understand how the code works, for example, by identifying dependencies and prerequisites clearly are the most salient purposes of using code examples.

Beyond demonstrating that code examples are used widely and for a broad range of purposes, the interviews also made clear that code examples are not helpful per se. Instead, they need to meet certain quality criteria in order to support the developers. First and foremost, code examples have to be programmed professionally. They are expected to show best practice approaches and to follow style principles accepted in the community.

> What's not good: if the samples are programmed badly. If I see, OK, that solves the problem, but in a bad way, violating other principles of professional programming. This should never be done. (Nicht gut ist es, wenn die Beispiele schlecht gemacht sind. Wenn man weiß, ok, das löst das Problem, aber es löst es auf so eine ‚dreckige'

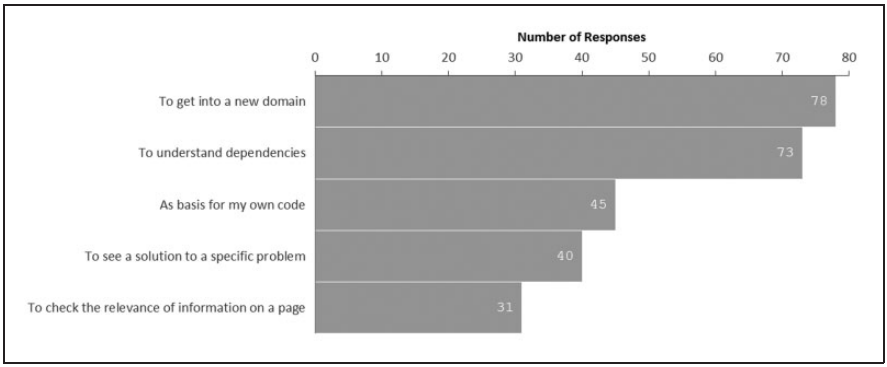

Figure 9. Frequency of answers selected in response to the question "For which purpose do you use code examples?" Multiple answers (up to three) possible per participant.

> Art, dass ich das wieder verwerfen muss, weil es gegen andere Grundprinzipien guten Programmierens verstößt. Das sollte man so auf keinen Fall machen, P13).

Several developers mentioned that they prefer code examples that can be understood without any explanation. However, they also noticed that this is often simply not possible and that in order to benefit from a code example, brief and informative explanations are necessary that explain what is going on and why the code looks the way it does.

> For example, a new API, and several artefacts come into play. You have to register something here and to import something there. There are complex dependencies. Some explanations are then helpful to understand how the individual pieces fit together. (Zum Beispiel eine neue API, da spielen verschiedene Artefakte mit rein. Man muss dort was registrieren, dort muss man was importieren. Es gibt komplexe Zusammenhänge - da wäre eine kurze Erklärung gut, wie die Einzelteile zusammenpassen, P03).

In addition, code examples are expected to be concise or—in case of more complex examples—be organized as a series of consecutive chunks. Moreover, to be helpful, they need to be complete and work correctly.

To identify typical problems that developers face when working with code examples, our questionnaire included the question "Which deficiencies do you encounter with code examples, and how often?" From the comments given in the interviews, seven answer options were derived. Respondents were asked to indicate whether and how often they have come across these deficiencies on a scale ranging from 1 (*never*) to 5 (*very often*). The results are shown in Figure 10.

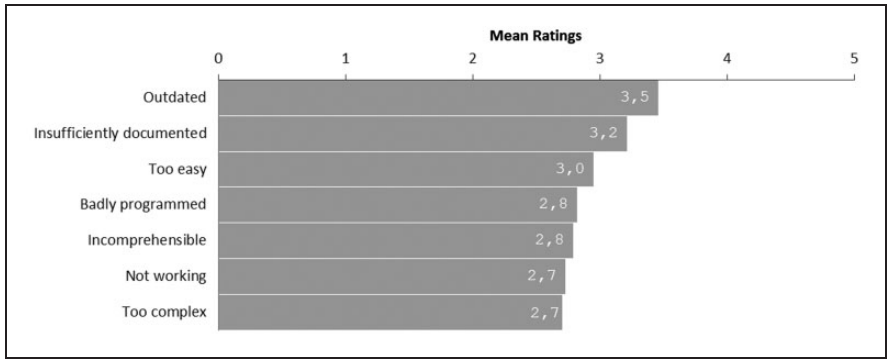

Figure 10. Mean ratings how often developers have come across the seven problems with code examples. Frequency of encounters was indicated on a scale from 1 (*never*) to 5 (*very often*).

As Figure 10 shows, the top issues with code examples seem to be that they are outdated or do not include sufficient explanation to make transparent how the code works.

General Discussion

Due to the growing importance of APIs, learning new APIs is a task that software developers today frequently face, but the task can be a challenge. This observation has generated considerable interest in the usability of APIs (Clarke, 2004; McLellan et al., 1998; Myers & Stylos, 2016). Besides API design and API tools, API documentation has been found to be a critical dimension with impact on API usability. However, prior research indicates that API documentation is a major obstacle in API learning and often does not appropriately serve the information needs of API developers during the learning process (Robillard, 2009; Robillard & DeLine, 2011). The main goal of the current study was to shed light on the issue which information developers want when they start to work with a new API, by contributing both qualitative and quantitative data that further our understanding of the initial learning goals developers formulate, the learning activities and strategies they adopt to accomplish these goals, and the information resources they use for learning.

To address learning goals and learning strategies of developers, we gathered data on questions developers raise to guide their learning efforts and the first steps they usually take at the beginning of the learning process. The interviews revealed that developers initially turn to an API with broadly two different goals in mind. One goal is to gather information that helps to decide whether the API may solve the problem that is at stake. Another goal is to actually learn to use the API, that is, to become able to actually solve that problem. Regardless

of which of these goals drives the initial contact with an API, developers first try to develop a high-level understanding of the overall purpose of the API and the main features it offers. For API documentation, the consequence can be derived that a brief description of the purpose and the main features of an API are needed. This brief overview needs to be easily accessible since this is the first piece of information that API developers will be looking for.

Once a high-level understanding of the API purpose and features has been formed, two different pathways seem to emerge that closely resemble the "systematic" and the "opportunistic" developer personas described by Clarke (2007) (see also Stylos, 2009). According to Clarke (2007), developers represented by the systematic developer persona work top down in the sense that they try to get a deeper understanding of the system as a whole before turning to individual components. On the other hand, the learning goals of opportunistic developers are more narrowly focussed on solving a particular problem and dependent on the specific issues and blockers they encounter while working toward a solution.

Our data reveal evidence for both these personas in terms of learning goals pursued and learning strategies adopted. Although learning goals are generally driven by particular tasks that need to be solved, some developers indicated that they want get going quickly and therefore are primarily interested in information on how to integrate the API and get it running, while other developers want to build a more thorough understanding of the API before addressing the specific tasks itself.

Consistent with the results regarding the initial questions developers raise when approaching a new API, our results suggest that developers adopt different learning strategies to dive into the API more deeply. One such learning strategy is code oriented and bottom up in the sense that developers want to start coding right away and therefore primarily look for information, such as code examples, that directly matches their problem or can at least serve as a promising point of departure. Other developers, however, proceed in a more top-down and concepts-oriented manner. Although these developers also have a specific problem or task that guides their learning activities, they start by learning important concepts or working through a "Getting Started" project or a beginner's tutorial, or by building a simple prototype themselves. An important issue that API documentation needs to address regardless of the learning strategy adopted is to provide clear entry points into the API and to make transparent how the API should be used to solve a specific problem (Robillard & DeLine, 2011).

With respect to information resources used, our results confirm observations from earlier studies that developers prefer to use search engines like Google to locate the information they need, and directly query the documentation offerings of an API supplier to a much lesser extent (Sillito & Begel, 2013; Stylos & Myers, 2006). API providers therefore need to make sure that their documentation offerings are visible on the web by making them accessible to search engines. The specific information resources developers look for depend on the

usage context. When starting to learn a new API, developers primarily look for code examples and tutorials whereas documents specifically addressing conceptual questions receive less attention. The API reference is an important in the initial learning stage as well, but gets even more important later on when developers start to work on their specific problems. Once developers have mastered the basics of an API, a learning resource that is also used rather frequently is developer forums like StackOverflow. In general, we also find evidence that developers seem to trust code more than documentation, supporting the view held by Maalej et al. (2014). However, our results also indicate that code examples should be presented as small chunks and along with comments that explain what the code is doing.

At a more general level, we conclude from our findings on learning goals, learning strategies, and information resources used that to successfully support the learning process, API documentation needs to serve the information needs of both the systematic and the opportunistic developer persona, for example, by not only providing information on API structure or the rationale behind API design decisions but also enabling developers to start work with code immediately. One possible strategy is to clearly separate code examples from text, for example—as some API documentation sites already do—by placing code examples in a separate column adjacent to the text. A particular challenge arises from our findings with respect to conceptual information that is necessary for any developer to use the API efficiently, such as conceptual background knowledge related to the specific domain covered by the API (Jeong et al., 2009; Ko & Riche, 2011). Conceptual knowledge like this needs to be presented redundantly and integrated into information resources that developers prefer to use, such as tutorials, code examples, and the API reference, to facilitate access for both the systematic and the opportunistic type of developer.

Regarding quality criteria developers apply to API documentation, our results show that high-level criteria like completeness and accuracy are very relevant to API documentation as well. Beyond that, special attention should be paid to defining a transparent information structure and to provide efficient navigation and search options, as developers often have problems finding and accessing the information they need. Taken together, our results on quality criteria support the view that API documentation should be regarded an asset of strategic importance. To satisfy the needs and expectations of the target audience, developing and maintaining this asset will have to involve the expertise not only of software developers, but also of information design and communication professionals.

Declaration of Conflicting Interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

Funding

The author(s) disclosed receipt of the following financial support for the research, authorship, and/or publication of this article: The work is supported by the German Federal Ministry of Education and Research (BMBF), FHprofUnt, grant 03FH014PX4 (to M. M.).

References

Alexander, K. P. (2013). The usability of print and online video instructions. *Technical Communication Quarterly*, *22*(3), 237–259.

Bogner, A., Littig, B., & Menz, W. (2009). *Interviewing experts* (Research Methods Series). London, England: Palgrave Macmillan.

Clarke, S. (2004). Measuring API usability. *Dr. Dobb's Journal*, *29*, S6–S9. Retrieved from http://www.drdobbs.com/windows/measuring-api-usability/184405654

Clarke, S. (2007). What is an end user software engineer? In *Dagstuhl Seminar Proceedings*. Schloss Dagstuhl-Leibniz-Zentrum für Informatik. Retrieved from http://drops.dagstuhl.de/opus/volltexte/2007/1080/

Dagenais, B., & Robillard, M. P. (2010). Creating and evolving developer documentation: Understanding the decisions of open source contributors. In *Proceedings of the 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (pp. 127–136). New York, NY: ACM.

Garousi, G., Garousi-Yusifoğlu, V., Ruhe, G., Zhi, J., Moussavi, M., & Smith, B. (2015). Usage and usefulness of technical software documentation: An industrial case study. *Information and Software Technology*, *57*, 664–682.

Guillemette, R. A. (1989). Usability in computer documentation design: Conceptual and methodological considerations. *IEEE Transactions on Professional Communication*, *32*(4), 217–229.

Hou, D., & Li, L. (2011). Obstacles in using frameworks and APIs: An exploratory study of programmers' newsgroup discussions. In *Proceedings of 2011 IEEE 19th International Conference on Program Comprehension* (pp. 91–100). Washington, DC: IEEE.

Hove, S. E., & Anda, B. (2005). Experiences from conducting semi-structured interviews in empirical software engineering research. In *Proceedings of 11th IEEE International Symposium on Software Metrics* (pp. 10–23). Washington, DC: IEEE.

Jeong, S. Y., Xie, Y., Beaton, J., Myers, B. A., Stylos, J., Ehret, R., & Busse, D. K. (2009). Improving documentation for eSOA APIs through user studies. In *International Symposium on End User Development* (pp. 86–105). Berlin, Germany: Springer.

Kim, M., Bergman, L., Lau, T., & Notkin, D. (2004). An ethnographic study of copy and paste programming practices in OOPL. In *Proceedings of International Symposium on Empirical Software Engineering* (pp. 83–92). Washington, DC: IEEE.

Knowles, M. S. (1975). *Self-directed learning: A guide for learners and teachers*. New York, NY: Cambridge Book Company.

Ko, A. J., DeLine, R., & Venolia, G. (2007). Information needs in collocated software development teams. In *Proceedings of 29th International Conference on Software Engineering* (pp. 344–353). Washington, DC: IEEE.

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

Kuckartz, U. (2014). *Qualitative text analysis: A guide to methods, practice and using software*. Los Angeles, CA: Sage.

Lethbridge, T. C., Sim, S. E., & Singer, J. (2005). Studying software engineers: Data collection techniques for software field studies. *Empirical Software Engineering*, *10*(3), 311–341.

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

Lutters, W. G., & Seaman, C. B. (2007). Revealing actual documentation usage in software maintenance through war stories. *Information and Software Technology*, *49*(6), 576–587.

Maalej, W., & Robillard, M. P. (2013). Patterns of knowledge in API reference documentation. *IEEE Transactions on Software Engineering*, *39*(9), 1264–1282.

Maalej, W., Tiarks, R., Roehm, T., & Koschke, R. (2014). On the comprehension of program comprehension. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, *23*(4), 31.

Mayring, P. (2014). *Qualitative content analysis: Theoretical foundation, basic procedures and software solution*. Retrieved from http://nbn-resolving.de/urn:nbn:de:0168-ssoar-395173

Mayring, P. (2015). *Qualitative Inhaltsanalyse: Grundlagen und Techniken* [Qualitative content analysis: Foundations and procedures] (12th ed.). Weinheim, Germany: Beltz.

McLellan, S. G., Roesler, A. W., Tempest, J. T., & Spinuzzi, C. I. (1998). Building more usable APIs. *IEEE Software*, *15*(3), 78–86.

Mihaly, F. (2011). Writing helpful API documentation [Blog post]. Retrieved from http://theamiableapi.com/2011/11/01/api-design-best-practice-write-helpful-documentation/

Myers, B. A., & Stylos, J. (2016). Improving API usability. *Communications of the ACM*, *59*(6), 62–69.

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). New York, NY: ACM.

Parnin, C. (2013). API documentation–Why it sucks [Blog post]. Retrieved from http://blog.ninlabs.com/2013/03/api-documentation/

R Core Team. (2016). *R: A language and environment for statistical computing*. Vienna, Austria: R Foundation for Statistical Computing.

Redish, J. C. G. (2000). What is information design? *Technical Communication*, *47*(2), 163–166.

Redish, J. C. G. (2010). Technical communication and usability: Intertwined strands and mutual influences. *IEEE Transactions on Professional Communication*, *53*(3), 191–201.

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

Robillard, M. P., & DeLine, R. (2011). A field study of API learning obstacles. *Empirical Software Engineering*, *16*(6), 703–732.

Schriver, K. A. (1997). *Dynamics in document design*. New York, NY: John Wiley & Sons, Inc.

Seaman, C. B. (2002). The information gathering strategies of software maintainers. In *Proceedings of International Conference on Software Maintenance* (pp. 141–149). Washington, DC: IEEE.

Shull, F., Lanubile, F., & Basili, V. R. (2000). Investigating reading techniques for object-oriented framework learning. *IEEE Transactions on Software Engineering*, *26*(11), 1101–1118.

Sillito, J., & Begel, A. (2013). App-directed learning: An exploratory study. *Proceedings of 6th International Workshop on Cooperative and Human Aspects of Software Engineering* (pp. 81–84). Washington, DC: IEEE.

Steinmacher, I., Chaves, A. P., Conte, T. U., & Gerosa, M. A. (2014). Preliminary empirical identification of barriers faced by newcomers to Open Source Software projects. In *Brazilian Symposium on Software Engineering* (pp. 51–60). Washington, DC: IEEE.

Stylos, J. (2009). Making APIs more usable with improved API design, documentation and tools (Doctoral dissertation). Carnegie Mellon University. Retrieved from http://www.cs.cmu.edu/~NatProg/papers/

Stylos, J., & Clarke, S. (2007). Usability implications of requiring parameters in objects' constructors. In *Proceedings of 29th International Conference on Software Engineering* (pp. 529–539). Washington, DC: IEEE.

Stylos, J., Faulring, A., Yang, Z., & Myers, B. A. (2009). Improving API documentation using API usage information. In *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing* (pp. 119–126). Washington, DC: IEEE.

Stylos, J., & Myers, B. A. (2006). Mica: A web-search tool for finding API components and examples. In *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing* (pp. 195–202). Washington, DC: IEEE.

Treude, C., Barzilay, O., & Storey, M. A. (2011, May). How do programmers ask and answer questions on the web? Nier track. In *Proceedings of 33rd International Conference on Software Engineering* (pp. 804–807). Washington, DC: IEEE.

Uddin, G., & Robillard, M. P. (2015). How API documentation fails. *IEEE Software*, *32*(4), 68–75.

Watson, R. B. (2015). *The effect of visual design and information content on readers' assessments of API reference topics* (Doctoral dissertation). University of Washington. Retrieved from https://digital.lib.washington.edu/researchworks/

Zibran, M. (2008). What makes APIs difficult to use? *International Journal of Computer Science and Network Security*, *8*(4), 255–261.

Author Biographies

Michael Meng is professor of applied linguistics at Merseburg University of Applied Sciences where he teaches courses on text analysis, text production and research design. Before joining university, he worked for 12 years as a technical communicator for an international software company. His research focuses on using empirical methods to study the effects of linguistic and design

variables on comprehension and the usability of information products in technical communication.

Stephanie Steinhardt earned a diploma in Technical Communication from Merseburg University of Applied Sciences. She teaches information design and also works as a freelance consultant and e-learning specialist for various clients.

Andreas Schubert received a Master's degree in Technical Communication from Merseburg University of Applied Sciences where he currently works as a research assistant in a project exploring strategies to optimize API documentation.