# The Role of Conceptual Knowledge in API Usability

Amy J. Ko
The Information School | DUB Group
University of Washington
Seattle, WA, USA
ajko@uw.edu

Yann Riche
Microsoft Corporation
Redmond, WA, USA
yannr@microsoft.com

***Abstract*—While many studies have investigated the challenges that developers face in finding and using API documentation, few have considered the role of developers' conceptual knowledge in these tasks. We designed a study in which developers were asked to explore the feasibility of two requirements concerning networking protocols and application platforms that most participants were unfamiliar with, observing the effect that a lack of conceptual knowledge had on their use of documentation. Our results show that without conceptual knowledge, developers struggled to formulate effective queries and to evaluate the relevance or meaning of content they found. Our results suggest that API documentation should not only include detailed examples of API use, but also thorough introductions to the concepts, standards, and ideas manifested in an API's data structures and functionality.**

***Keywords —API usability, documentation, feasibility***

## I. Introduction

In today's consumer software marketplace, software designers have a wide range of platforms they may choose from to develop new applications. For example, mobile application developers can choose from iOS, Android, Windows Mobile 6.5, Windows Phone, Symbian, HTML5, among many other niche platforms, weighing the strengths and weaknesses of each platform's technical capabilities, market potential, technical support community, user community, among other factors.

While developers may choose a platform for a variety of reasons, one important factor in this decision is assessing the *feasibility* of implementing particular features on a platform. For example, suppose a developer has an idea for a real-time augmented reality game, but to make the game work, the platform needs to be able to retrieve at least 2 frames from camera sensor per second in good light at a resolution of 640 x 480. With the amount of content online concerning these platforms, many feasibility assessments can be quite simple. For example, a Google search for "iPhone 3G camera resolution" reveals a MacRumors forum thread in which someone answers this exact question and cites (a now outdated) technical specification for the iPhone 3G.

Not all feasibility assessments are so simple, however. This is particularly true when the behavior desired for an application involves conceptual knowledge that may be new to a developer. For example, in researching the sensor speed of the iPhone 3G camera, one discovers that searching for and understanding documentation about mobile image capturing requires a great deal of knowledge of ISO speeds, exposure time, and the role of lighting conditions, and, among other things, the capabilities and limitations of Apple' iOS *UIImagePickerController* class. To learn all of this, developers must search, browse, read, and synthesize information from a wide variety of sources, including API documentation, tutorials, example code, and community-generated content.

While prior work has considered many aspects of API usability and documentation [1,3,5,6,7], few have investigated how developers' conceptual knowledge about a requirement influences the success of their interactions with API documentation. To investigate this influence, we designed a study in which software developers researched the feasibility of two requirements concerning Bluetooth and wifi protocols, for both Windows Mobile 6.5 and Android 2.1 platforms. By analyzing developers' utterances during these tasks, we contribute evidence that not only was conceptual knowledge essential in helping developers make sound judgments about the feasibility of the requirements, but it was a basic part of formulating effective queries and evaluating the relevance of search results and API documentation. These findings have several implications for the design of API documentation and software development Q&A sites.

## II. Related Work

Prior studies have explored many activities related to searches for unfamiliar content. For example, research on information foraging theory has been applied to accurately predict where users will navigate based on keywords on pages they visited previously [2]; this work suggests that, at least for informational searches within a single site, that the intent of what people search for can be partially captured by what phrases they choose to navigate. Other research in information science has long studied the question negotiation process in information seeking, distinguishing between actual but unexpressed information needs, the conscious understanding of the need, verbal statements of a need, and questions as stated to an information retrieval system (whether a search engine, index, librarian, or friend) [8]. In this process, the information seeker often does not know what knowledge they lack and therefore cannot formulate an effective query without first learning about what information is available (for example, through an index or catalog).

These basic observations about information seeking have been explored in many software development contexts, revealing several important factors in the design of online API documentation. Studies have shown that the most desired and effective way of conveying this knowledge is through annotated code examples [1,3,5,6,7], that developers must engage in a series of query reformulations to identify the appropriate platform-specific terminology that indexes a behavior [1,5,6], and that the legitimacy of sources [3] and visual design of the web sites [3,5] can be important factors in developers' decisions to use an online API resource.

Only a few studies address the role of conceptual knowledge in API usability. Some studies have reported that developers find detailed introductions to a platform's architecture and design are critical [5,6,7]; these studies do not, however, investigate *why* such overviews are important. Another recent study highlighted the general importance of developers' background in understanding API resources [7]. This prior work makes it clear that conceptual knowledge is important; our study seeks to understand why it is important.

## III. Method

Our study asked software developers to assess the feasibility of two requirements for two different mobile platforms across four 20-minute sessions. Testing two platforms allowed us to better isolate whether difficulties were due to the online resources or a lack of conceptual knowledge. The two requirements we presented were described in a scenario involving a company that wanted to design a mobile application that would collect data about the use of various Bluetooth protocols in a city and securely transmit the information over a wifi network. Participants were asked to imagine that they were hired by the company to decide whether to use the Windows Mobile or Android platform to implement this application. We worded the requirements to avoid revealing important names in the API for either platform, using only industry-wide named standards. The two requirements given to participants were:

***Requirement****: The application must be able to obtain a list of discoverable Bluetooth-enabled devices in its proximity.*

***Requirement****: The application must be able to determine whether the wifi network the phone is connected to is using a WPA or WEP secure connection.*

The actual feasibility of these requirements was nuanced: the first requirement was feasible, but could only be accomplished over a period of time that was not under the developer's control; the second requirement was only feasible on devices that provided detailed device information in a configuration string exposed by the API.

We recruited 7 developers, including senior undergrads and masters students who had returned from industry (5 male, 2 female). All reported working on large software systems. When asked to indicate their experience with 41 languages and APIs, the number ranged from 5 to 14, with a median of 10. The most frequent responses were C, C++, C#, Java, JavaScript, and PHP. All were students majoring in either CS, computer engineering, or information science. Only two described having significant experience with either Bluetooth or wifi protocols. None had written applications for either mobile platform, but most had visited Microsoft Developer Network (MSDN).

The study procedure was as follows. Participants were brought into a lab and told they would be assessing the feasibility of two software requirements for each platform. After a 5-minute think aloud training session involving a non-software development related search task, participants were given 20 minutes to assess the feasibility of each requirement/platform pair. The order of these pairings was counterbalanced to minimize the impact of learning effects on our analyses. Participants were allowed to use any resource on the Internet that they could find or download.

To understand the relationship between conceptual knowledge and participants' research, we asked participants to provide a verbal judgment of the feasibility and difficulty of the requirement they were working on before, during, and after each pair of requirements and platforms. Participants were prompted regularly approximately every 5-10 minutes for their current assessment. At the end of each session, participants were asked to state their final assessment, identifying evidence that they had found to support their assessment. We screen captured and audio recorded each session.

## IV. Analysis

In designing our analysis, we considered grounded theory, but felt there was enough research on API understanding that it would have been inappropriate to start without a theory of what the data contained. Our approach instead explicitly focused on declarative judgments in participants' utterances, which are the basis of many theories of information seeking (e.g., [8]). We therefore operationalized utterances as complete sentences that were separated in both *time* and *topic*. After applying this definition, our resulting data set had 2,633 utterances.

Next, we performed open coding on these judgments to arrive at the judgment categories, applying selective coding on these judgments on the original transcript data. We operationalized these judgments as declarative sentences in which the participant stated a claim about an information resource (e.g., "Windows CE which is not what we want.") or themselves (e.g., "Actually, I don't know how to start"). Non-declarative utterances describing a participant's actions (e.g., "I'm going to check this link,") were not included in our analysis. Our final 5 judgment categories were:

- **Relevance** judgments stating whether the participant believed the information would inform the assessment task (e.g., "*Nothing is on the page*").
- **Usability** judgments concerning the predicted or actual experience of navigating a resource (e.g., "*[This JavaDoc] documentation will be bad.*").
- **Audience** judgments concerning for whom the participant believed the resource was designed (e.g., "*I think this is for people who already know Android development*").
- **Proximity** judgments about nature of the search space being explored and the participant's location in it (e.g., "*Um... okay... it's getting colder and colder.*").
- **Metacognitive** judgments concerning a participant's ability to perform the task (e.g., "*Maybe there is a problem with my method of searching.*")

To measure the inter-rater reliability of our coding scheme, the two authors redundantly coded one participant's utterances (accounting for 6% of utterances), reaching 86% agreement on whether an utterance was a judgment and 82% agreement on selecting one of the five kinds of judgments listed above. With this level of agreement reached, the first author then coded the remaining utterances. These categories were then used to focus our qualitative analyses only on statements of particular kinds.

## V. RESULTS

By far, the most salient characteristic of the participants' work was the variation in the substance and depth of their assessments. Some participants only managed to arrive at vague conclusions, informed by assumptions and expectations rather than evidence:

*"There was a little bit of wifi code, but it was very scattered and then… there's someone mentioning they were doing this sort of thing which made me sort of, it implies that it's doable, but... the amount of information I was able to dig up was... not helpful at all.*

Other participants describe in detail the code they would write to address a requirement, including its limitations:

*"It was startDiscovery()… it can scan for 12 seconds and retrieve a list, or start sending back essentially representations of the devices, it was like: oh! That's handy… The only thing that I'm worried about at the end is the admin privileges on the phone for bluetooth seem like when you're running the application, unless you have those, you won't be able to do anything."*

In analyzing the sources of this variation, many previously reported factors were involved: all participants focused on finding code examples [1,3,5,6,7], but struggled to find the platform-specific terminology needed to retrieve them. To identify concept terminology, participants reformulated their queries, determining which words uniquely indexed the information they were looking for [1,5,6]. When participants found overviews of platform architectures, they reported finding them helpful in knowing what to search for and whether they were making progress [5,6,7]. Participants also were hesitant to read sites that had cluttered or inconsistent visual design, focusing primarily on sites that appeared to be easy to read and browse at first glance [3,5]. The Windows Mobile documentation posed several usability problems that prevented participants' progress; many participants struggled to even find the MSDN API documentation, let alone browse it, because much of the content concerning Windows Mobile was not clearly versioned or explicitly intended for developers.

In addition to replicating the results above, however, we also found participants' conceptual knowledge of the Bluetooth and wifi protocols greatly impacted their ability to find and understand API documentation. Because of space limitations, we will not discuss the trends in the first four kinds of judgments, as they largely replicate the findings in previous work [1,3,5,6,7]. We instead focus on the *metacognitive* judgments, in which participants made statements about their own ability to make progress on the feasibility assessments. In particular, participants discussed, unprompted, the importance of conceptual knowledge in two aspects of their search for information about each platform and requirement.

### A. *Conceptual Knowledge Enables Effective Queries*

One of the key differences between developers who arrived at detailed assessments and those who did not was the specificity of their initial queries. Developers who only arrived at vague assessments began with terse, imprecise queries, in some cases clearly extracted from the terminology used in the requirements given to them such as "bluetooth android", "bluetooth android api", and "windows mobile 6.5 wifi". These developers knew their searches were vague, but wanted to see what kinds of information existed about the platform before deciding what to search for. These queries did lead to content intended for developers, but often only provided vague overviews, rather than details. Developers struggled to find what terminology by might uniquely index content related to the requirement:

*"…maybe just I cannot find the right keywords."*

*"I kind of wish I had seen something more like, reference. API. Something that could give me a set of keywords I could google."*

Developers who struggled to find the right keywords eventually found effective queries:

*"Okay, so at this point I feel like I know the keyword I need to know, such as action discoverable, and, you know BluetoothAdapter."*

*"I see the keyword I want multiple times, in logical progression, right? To me its sounding like yes it can be done, it's not too hard. Yeah?"*

Despite their success at finding relevant keywords, participants expressed their desire for materials that explained the conceptual knowledge they felt they lacked:

*"I'd like a general overview of how Bluetooth connections work. I know practically nothing about them so that would have helped a lot. And, maybe an index into the page that would help to give me more abstract terms for how to think about it."*

*"For me, I would like to see the architecture, a simple architecture to introduce Android bluetooth, how they work, pair. Then I will get a sense to this whole environment, how they work, and I will see, see how to enable this bluetooth in android, and then I would like to see sample and demo finalized, and then finally I would like to see the code."*

In contrast to the participants who only arrived at vague and inconclusive assessments, the participants who arrived at detailed and conclusive assessments knew immediately what terminology to use in their initial queries. One participant, for example, began with the query "bluetooth android api discoverable"; this led directly to an Android code snippet that demonstrated how to obtain a list of the discoverable Bluetooth devices in range.

### B. *Conceptual Knowledge Enables Relevance Judgments*

In addition to being essential to forming effective queries, conceptual knowledge about the Bluetooth and wifi protocols were also essential to participants' ability to judge the relevance of the content they found online. For example, developers often arrived at pages they thought might be relevant, but struggled to know how to proceed:

*"I'm not sure what I'm looking at exactly. It looks like a wifi page for android. I guess I don't know what an insecure network would look like in terms of its wifi configuration."*

*"I think I just mostly feel hampered that I just don't know the network protocols as much. I think that's the biggest thing."*

In many cases, the developers' indecision stemmed from their unfamiliarity with particular terminology:

*"'put radio into discoverable mode'. What is radio? I'm not quite sure here what radio means."*

To resolve this conceptual confusion, 6 out of the 7 developers went to Wikipedia and other encyclopedic resources in order to help them interpret the API documentation they were trying to understand:

*"I'm scanning over things and looking for things about networks and connections on Wikipedia."*

Encyclopedic resources were rarely found helpful, however, because the information was not described in the context of the platform and how it manifested the protocols.

Ultimately, these developers felt they spent much of their time not knowing they were on the right page:

*"Since I don't have this kind of knowledge, it's kind of hard for me. I would just try to put some keywords, it's pretty much what I know from this mobile device, so when I put it, if I cannot direct to right page, then I'll spend a lot of time on the wrong page, keep searching wrong information, or maybe the right page but I didn't get it."*

In contrast to participants who lacked the conceptual background in Bluetooth and wifi security, participants who succeeded in making more substantial feasibility assessments could not only recognize pages as relevant, but also extract software architectural knowledge from them:

*"Oh, I see, it's an asynchronous call, that's nice, so it will just start discovering and you'll be notified immediately as each one is found. And then at the end of that you could take the whole list and e-mail it back, or whatever you were going to do."*

In situations like these, participants immediately recognized terminology, standards, and acronyms that were fundamental and unique to the network protocols. Even for these developers, however, code examples were not enough: even after finding them, they searched for walkthroughs about the design and rationale for a snippet, so they could make predictions about the feasibility of variations on the source code.

## VI. Discussion and Conclusion

As with any empirical study, ours has several limitations. By asking participants to reflect regularly on their feasibility assessments, we may have altered the nature of their search strategies or compelled them to reflect more than they would search on their own. The search sessions also imposed an artificial time limit on the assessments, which may not occur in more ecologically valid settings. The participants were also all students and had not worked as professional software developers. Developers with such inexperience may not usually be responsible for making feasibility assessments.

With these limitations in mind, our results suggest that a fundamental prerequisite for finding and effectively using API documentation is having the conceptual knowledge necessary to (1) identify relevant, unique terminology for searching, and (2) evaluate the relevance of content found online that uses the terminology. This differs from prior work on the usability of API documentation, in which developers often had a clear conceptual understanding of the behavior they are trying to implement, but simply could not remember the particular name of a function or how to call it [1].

The implications of these results are many. First, documentation should be designed not only for developers with significant background in the concepts used in an API, but also for developers who are unfamiliar with them. For example, in the context of Bluetooth and wifi protocols, documentation could include in-depth tutorials that not only provide overviews of the protocols, but how the protocols are manifested as classes, methods, functions, and data structures. It was particularly important in our study that the materials are written concretely, in terms of the platform's implementation of the standards and protocols and not in general terms.

Our results also suggest that one particularly useful introduction to a platform and its concepts would include a list of the major terminology used throughout the application and what the terminology means. The developers in our study who were unfamiliar with networking protocols were essentially using Google as a glossary, extracting terminology from Q&A forums for later use in queries. API documentation might be greatly improved by providing proper glossaries, with linked tutorials introducing each major concept and how it is used on the platform to implement functionality.

## Acknowledgments

We thank Jeff Stylos, Jeanine Spence, and the rest of the Microsoft developer division UX team who helped design the study. This material is based in part upon work supported by the National Science Foundation under Grant Number CCF-0952733. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

## References

[1] Brandt, J., Guo, P.J., Lewenstein, J., Dontcheva, M., Klemmer, S.R. (2009). Two Studies of Opportunistic Programming: Interleaving Web Foraging, Learning, and Writing Code. *ACM Conference on Human Factors in Computing Systems*.

[2] Chi, E.H., Pirolli, P., Chen, K., and Pitkow, J. (2001). Using information scent to model user information needs and actions and the Web. *ACM Conference on Human Factors in Computing Systems,* 490-497.

[3] Dorn, B. and Guzdial, M. (2010). Learning on the job: characterizing the programming knowledge and learning strategies of web designers. Conference on Human Factors in Computing Systems, 703-712.

[4] Ko, A.J., Myers, B.A., and Aung, H.H. (2004). Six Learning Barriers in End-User Programming Systems. *IEEE Symposium on Visual Languages and Human-Centric Computing*, 199–206.

[5] Myers, B.A., Jeong, S.Y., Xie, Y., Beaton, J., Stylos, J., Ehret, R., Karstens, J., Efeoglu, A., Busse, D.K. (2010). Studying the Documentation of an API for Enterprise Service-Oriented Architecture. *The Journal of Org. and End User Computing*, 22(1), Jan-Mar, 23-51.

[6] Nykaza, J., Messinger, R., Boehme, F., Norman, C.L., Mace, M., and Gordon, M. (2002). What programmers really want: results of a needs assessment for sdk documentation. *International Conference on Computer Documentation*, 133–141.

[7] Robillard, M.P. and DeLine, R. (2010). A Field Study of API Learning Obstacles. *Empirical Software Engineering,* 1382-3256.

[8] Taylor, R.S. Question-Negotiation and Information Seeking in Libraries. *College & Research Libraries, 29*(3), 1968, 178-194.