Empirical Software Engineering 2025 30:73

Evaluating Interactive Documentation for Programmers

Mathieu Nassif · Martin P. Robillard

Received: date / Accepted: date

Abstract Documentation for software programmers is often delivered online. Yet, its format seldom leverages interactive features supported by web technologies. Researchers have proposed techniques to include interactive elements in documents. We investigated one approach, Casdoc, to understand how programmers adapt their navigation behavior to interactive documents. During the study, participants completed programming tasks that involved an unfamiliar library. They used documents that presented information in two formats: one interactive and one traditional. We analyzed recordings of the participants' usage of the documents and their answers to a post-study questionnaire to identify strengths and limitations of the two formats. We found that the interactive format helped find answers rapidly for short factual queries, whereas the traditional format was better suited to open-ended queries that required synthesizing more information. Overall, our results show the potential of leveraging alternative formats, even within the same document, to address a broader spectrum of information needs.

Keywords Interactive documentation · Programming tutorial · Reading patterns · Web-based documents

1 Introduction

Programmers can find a large amount of online resources to learn about unfamiliar software technologies, their application programming interfaces (APIs), and related programming concepts. These documents can take many forms, including official documentation from the developing organization, public forums such as Stack Overflow, community-maintained knowledge bases such as Wikipedia, and a plethora of tutorials and blog posts. Despite this wide range of documentation types, most

This work was supported by the Natural Sciences and Engineering Research Council of Canada and the Fonds de Recherche du Québec – Nature et technologies.

M. Nassif and M. P. Robillard
School of Computer Science, McGill University, Montréal, Canada
E-mail: mathieu.nassif@mail.mcgill.ca
E-mail: robillard@acm.org

documents present information in a rigid structure: a fixed sequence of headers, paragraphs, and supporting elements such as code examples, figures, and tables. This rigid structure is not designed to adapt to the navigation patterns of programmers. Studies have found that programmers often jump between sections of a document, focus on elements such as code examples, and try to only read the content relevant to a specific task (Hornbæk and Frøkjær 2003; Brandt et al. 2009).

Different formats have emerged alongside text-based web pages that change how programmers interact with documentation, such as video tutorials (van der Meij and van der Meij 2014; MacLeod et al. 2017; Moslehi et al. 2022). New approaches to create (Chi et al. 2012) and consume (Khandwala and Guo 2018) video-based content demonstrate the potential of this format to help programmers exchange knowledge. Augmented reality (AR) and virtual reality (VR) environments also provide a new way to explore and interact with representations of software systems (Chimalakonda and Venigalla 2020; Fittkau et al. 2017). Nevertheless, text-based documents have several benefits. They are familiar to most writers and readers, they are supported well by web browsers and content management systems, and they do not depend on specialized software products or platforms. Thus, many approaches focus on automatically extracting information from large text corpora based on programmer's queries (da Silva et al. 2020; Wu et al. 2023). Other techniques can produce the initial query from the source code context in an integrated development environment (Ponzanelli et al. 2014). Other approaches synthesize the content of documents, e.g., based on programming tasks, to help programmers locate more efficiently the information they need (Treude et al. 2015; Sun et al. 2019). Recently, interactive systems that support natural language queries, including platforms such as ChatGPT and Copilot, offer an alternative mode of interaction with documentation (ter Hoeve et al. 2020; Todi et al. 2021; Xiao et al. 2024; Tomova et al. 2024).

In a prior publication, we introduced Casdoc, an interactive format for code examples (Nassif et al. 2022). We designed Casdoc to explore an alternative information presentation approach for creating text-based documents that are inherently adaptable to the readers' needs. Casdoc uses a complete code example that captures the purpose of a document as its base. It adds additional information to this code example as a graph of concise fragments that readers reveal on demand. The implementation of Casdoc allows readers to navigate this graph through interactive dialogs and popovers. Thus, Casdoc documents can include a large amount of information in a deep structure of nested fragments that annotate the code example. We refer to these interactive fragments that contain explanations of the code example as *annotations*.

The diversity of approaches to present and consume documentation illustrates several compromises between two desirable qualities: More interactive interfaces can better adapt the information to the needs of a programmer, while less interactive interfaces require fewer actions to access the information. This trade-off between adaptability and navigation costs may influence programmers to prefer one type of documents over another based on inherent or contextual factors. Understanding these factors can help create or retrieve documents that better satisfy the needs of their readers.

We report on a user study to investigate how programmers adapt their within-document navigation behavior to an interactive document format. We sought to understand the benefits and limitations that an interactive format such as Casdoc can have, compared to a non-interactive format, when used to help readers

find the information they need. Thus, we elicited *qualitative navigation patterns* that programmers use when working on a given task using either Casdoc or a traditional format. These patterns can help inform the design of new formats or improve existing ones. We designed the study to answer two research questions:

1. In which situations does a programmer choose to use the interactive format vs. the traditional format?
2. How does the interactive format support or interfere with a programmer's actions with the documentation?

The study consisted of asking participants to solve a series of implementation tasks using an unfamiliar application programming interface (API) during a one-hour session. Participants had only access to a limited set of documents that we authored specifically for the tasks. Each document combined both formats, showing a Casdoc code example at the top followed by all of its interactive content replicated as non-interactive text below the code example. We analyzed the video recordings of the participants' use of the documents during the sessions to identify recurring navigation patterns.

We found that neither format, on its own, is sufficient to properly support all navigation patterns, but that the two formats complement each other well within a document, even if information is duplicated as a result. We also observed that navigation patterns depend not only on the *personal preferences* of a reader, but also on the *type of information* sought and the *context of the search*. For example, when searching for a specific piece of information about a known code element, rapidly locating the fragment that contains this information is desirable. However, when searching information about an unfamiliar concept, spending more time perusing related information is preferable to locating only, e.g., a definition of the concept.

Although the study was specific to the Casdoc documentation format, the results provide new insights into programmers' interaction with documentation, which can help writers design new software documentation formats. In particular, as tutorials increasingly include dynamic navigation features, our study provides an empirical basis to evaluate and improve these features. Our study also highlights the potential benefits of multimodal formats and the varying importance of different document properties (e.g., conciseness) given the reader's programming task.

Online Appendix

This work is complemented by a publicly available online appendix (Nassif and Robillard 2024a). The appendix includes all the material necessary to replicate the study, including the document set, task instructions, and configuration files for two integrated development environments. It also includes the annotation guidelines and results of each phase of our analysis (see Section 3.4). Additionally, readers can find documents in the Casdoc format at https://codesample.info/.

2 Background and Related Work

Effective documentation is a crucial, yet often lacking, aspect of software development (Robillard and DeLine 2011; Aghajani et al. 2019). Our study contributes to the

research on the behavior of readers interacting with documents, as well as to innovations in the design of online software documentation. It investigates a question at the intersection of the software engineering (SE) and the human-computer interaction (HCI) domains, as we seek to improve the user interface of documents to help software programmers.

We scope our work to text-based documents read on a typical computer monitor, as opposed to video resources and AR/VR environments. Additionally, our work focuses on the *structure* of information within a document and the *interaction features* to access this information. Although other aspects of a documentation format can have an important impact on its quality, such as *aesthetic* decisions, we exclude such considerations from our investigation.

2.1 Interacting with Text-Based Documents

Various studies investigated what information programmers seek in documentation (Bouraffa and Maalej 2020). Sillito et al. (2008) and Duala-Ekoko and Robillard (2012), for example, observed participants during software maintenance and implementation tasks, respectively, to identify questions asked by programmers. They synthesized their results into 44 and 20 typical questions, respectively. Many studies analyzed public online documentation, such as API reference (Maalej and Robillard 2013) or Stack Overflow posts (Liu et al. 2021b; Beyer et al. 2020), to elicit types of information available to their readers. The outcome of these studies can serve as a valuable framework to understand the information transfer between API developers and users, for example by differentiating between functional, structural, uninformative, and other types of content in API documentation, or between questions about API usage, unexpected behavior, version evolution, and others on Stack Overflow.

Aside from studies about *what* programmers look for in documents, studies about *how* they look for information can guide the improvement of documentation. Prior work has characterized programmer search activities in their code editor and documentation during observation studies. For example, they showed differences in information searches during debugging and implementation tasks (Ko et al. 2006), and identified recurring challenges in the use Wikipedia articles as learning resources on complex computing concepts (Robillard and Treude 2020). Artifacts generated by users of popular development platforms can also reveal insights into the search behavior of large populations. For example, a study of search query logs on Stack Overflow highlighted several patterns, e.g., related to how programmers reformulate unsuccessful queries (Liu et al. 2021a).

In the HCI community, researchers also reported on studies of how readers interact with online documents, although not necessarily about software programming. For example, Hornbæk and Frøkjær (2003) studied navigation patterns of students reading scientific articles. In particular, they compared three formats: a classic "linear" format, a "fisheye" format that allows readers to expand and collapse parts of a document, and an "overview+detail" format that shows a smaller version of the document in a left pane. They observed that the fisheye format, which shares some conceptual similarities with Casdoc, led to faster task completion times, but also to more inaccuracies in the participants' answers. Miniukovich et al. (2017) synthesized guidelines from the literature for increasing the readability of web documents, with a special focus on accommodating dyslexia. Chen et al. (2012) studied how readers interact with

documents across multiple types of devices, such as desktops, tablets, and e-readers. They proposed a novel system to increase the synergy between devices, but also noted emerging interoperability challenges, such as a practical limit on the number of devices. Research on *active reading* also revealed common document interaction patterns that readers use in professional settings and suggested new approaches to support them (Hinckley et al. 2012). For example, Tashman and Edwards (2011a) identified expectations and limitations of document reading software through a diary study, interviews, and participatory design workshops. They used their results to design active reading features (Tashman and Edwards 2011b). Such studies help contextualize our research on the use of software documents for development tasks within the larger scope of information seeking patterns.

2.2 Innovative Documentation Formats

The explosion of online documentation created an opportunity to explore innovative solutions to improve the formats of documents, now unconstrained by the physical limitations of printed resources. For example, web documents can include collapsible sections, hyperlinks between different fragments, and tabbed containers showing equivalent code examples in alternative languages. Some documents include more complex features, such as runnable code examples that programmers can edit (Miller and Ranum 2012). For example, the CSS reference documentation on Mozilla's MDN Web Docs often includes editable CSS rules that programmers can try themselves.[1] Pushing the idea further, Guo (2013) designed a tool to visualize step-by-step executions of programs for learning purposes. The web-based tool includes a representation of the program's memory to help programmers understand and reason about the operations in the code.

Many novel formats focus on the visualization of and interaction with machine learning (ML) models (e.g., Mellis et al. 2017; Ye et al. 2021). For example, Bäuerle et al. (2022) designed Symphony, a technique to create interactive interfaces to document machine learning models. Symphony employs reusable components that allow users to edit data sets and analyze properties of the different models within the document. Studies of interactive interfaces for machine learning models showed that they can ease communication between different groups of stakeholders (Feng et al. 2023) and improve the performance of optimization algorithms in complex situations (Higuchi et al. 2021).

Aside from machine learning models, researchers have also studied techniques to help readers interact with data presented in documents through tables (e.g., Kim et al. 2018) and figures (e.g., Heer et al. 2005). Techniques such as Chameleon (Masson et al. 2020) and Charagraph (Masson et al. 2023) can overlay supplemental interactive figures on static images or generate interactive graphs to visualize data described in non-interactive documents, respectively. Others proposed to generate concise facts about data presented in documents (Wang and Kim 2023). Studies of these formats revealed that they encourage readers to engage more with the documents, but also revealed design challenges, such as the personalization of the new features to the reader.

[1] See, for example, the top of the page about the `transition` property: https://developer.mozilla.org/en-US/docs/Web/CSS/transition

Fig. 1: Code Example with Casdoc Annotations

Closer to the focus of this article, other formats target code fragments to add information inferred from multiple documents. For example, color highlights, distribution charts, and interactive selections can help programmers compare code examples from alternative libraries (Glassman et al. 2018) or synthesize large sets of code examples (Yan et al. 2022) to accomplish a target task. These interactive visualizations helped participants explore more code examples and find more accurate information than with traditional documents.

Another closely related work is that of Horvath et al. (2022). They designed and implemented a new approach, Adamite, to create and exchange text annotations of web documents. Whereas Casdoc annotations are created as part of a document by its author, Adamite supports user-generated annotations. These annotations can help readers keep track of their thoughts when learning to use a new API and identify ambiguous or incorrect fragments of the documents.

2.3 The Casdoc Format

Casdoc is a novel format for web documents that allows document writers to augment Java code examples with embedded explanations, and that defines a protocol for readers to interact with these explanations (Nassif et al. 2022; Nassif and Robillard 2024b). The natural language explanations are presented as concise fragments called *annotations*, not to be confused with "annotation types" that are part of the Java language. Although Casdoc currently only supports Java code examples, its design principles could be applied to other languages.

Figure 1 shows a sample document using this format. The objective of Casdoc is to provide an alternative to documents structured as a linear sequence of information fragments. Thus, in a Casdoc document, annotations are the primary way to both organize and access a document's content. This strategy contrasts with other formats that use interactive features, such as dynamic tables of contents, hyperlinks, and collapsible sections, as additional support to help readers navigate within a document. Annotations are also already used to present secondary information in web documents, resulting in isolated interactive elements in the otherwise linear flow of a document's content. In contrast, we designed Casdoc to create documents that consist mainly of a single code example with minimal text outside annotations.[2]

When opening a document, only the code example is visible. Visual markers (i.e., underlines and margin brackets) identify the elements that have additional

[2] The results of our study, however, showed that a more nuanced approach may be desirable to support varied information search scenarios.

explanations in an annotation. Readers can temporarily reveal a floating annotation by hovering over its corresponding term in the code example. Readers can also pin annotations to the document by clicking on the term, then resize and move them to organize the information on the page according to their needs. Annotations can be nested to arbitrary depths, creating a hierarchical structure of the content. A search bar allows readers to search for text queries within all annotations, including those that are nested or hidden. We refer to floating annotations as *popovers* and to pinned annotations as *dialogs*, consistently with the graphical elements used to implement the format. The example document in Figure 1 shows one pinned dialog (center) with a nested floating popover (right).

The intention behind the design of Casdoc is to allow programmers to find information about code examples by following an explicit structure, instead of searching through large amounts of text. By hiding most of the content of a document, the format allows content-heavy documents to appear less verbose.

The focus on code examples as the initial view of a document is consistent with prior work, which demonstrate the value of code examples in programmer documentation (Brandt et al. 2009; Head et al. 2018). Contextualized annotations can help programmers understand the subtleties, variation points, or decision rationales of the code.

In contrast to some prior work, Casdoc is not a solution to allow readers to modify the content of a document. Rather, readers only modify how the (fixed) content is presented, by revealing, hiding, and moving annotations. Casdoc is also not a solution to automatically generate code annotations based solely on the initial code example (such as, e.g., Head et al. 2015; Suzuki et al. 2017; Hu et al. 2020). The document writer must create the annotations. Casdoc supports only one source of automatically imported annotations: API reference documentation for library types and members.

We reported on a field study in the prior work that introduces Casdoc (Nassif et al. 2022; Nassif and Robillard 2024b). During the field study, over 300 students used Casdoc documents for several months as part of their course material. The documents included additional client-side scripts to log all interactions with Casdoc's features. We analyzed the generated logs to evaluate how often different features were used. We found, for example, that participants interacted with nested annotations as often as with top-level annotations, but that the additional actions required to reach nested annotations makes them less suitable for important information. Although the field study had a large scale and realistic conditions, its design did not support providing insights about the motivations and thought processes of participants when they used specific features. The laboratory study reported in this article addressed this limitation by directly observing participants when they use Casdoc.

3 Study Design

We asked participants to perform several programming tasks that involved an unfamiliar API using a set of documents we provided. All participants were in the same experimental condition, using documents that combined Casdoc-formatted and non-interactive text to present information. While participants were working on the tasks, we recorded their screen and audio to analyze how they sought information and navigated the content of the documents.

The goal of the study was to explore how well an interactive format supports programmers looking for information, rather than to demonstrate the superiority of any single format. This goal motivated an in-depth analysis of fewer participants over a statistical analysis of performance measures, such as task completion times or error rates, with a large sample from a target population. Additionally, we scoped the study to the navigation patterns within a single document, as opposed to the synthesis of information from multiple sources.

The study focused on the interactivity aspect of the documents and its impact on information seeking activities. Interactivity constitutes an interesting trade-off in the design of a format: Interactive documents can better adapt to the needs of their readers, but at the cost of additional actions to reach relevant information.

3.1 Study Environment

We recruited 13 computer science students (seven undergraduates, five Master's, and one PhD) from our university to participate in the study. Participants had from one to ten years of experience programming in the Java language (average: 3.5 years). Five participants identified as male, seven as female, and one preferred not to disclose their gender.

In this study, students were not a proxy for the target population. All participants were programmers who used online documentation to learn programming knowledge. Thus, we recruited students as a subgroup of the population of programmers who need to learn about unfamiliar APIs.

Each study session took place during a videoconference with one of the investigators. Participants were compensated at the end of their session.[3] The same investigator conducted all sessions.

Each session started with a short pre-questionnaire in which participants self-reported their familiarity with SQL and JDBC, followed by a pre-recorded video of the study instructions and the format of the documents. The investigator then asked the participants to locate a specific piece of information within a training document. The objective of this training activity was to help participants familiarize themselves with the documentation format. The training session involved a topic unrelated to the programming tasks.

The participants then worked on the programming tasks for 40 minutes. We asked participants to comment aloud on their information search processes, such as the information they looked for, as they worked on the tasks. The investigator would occasionally prompt participants who remained silent for too long. After 40 minutes, participants had the option to continue working on the tasks for up to an additional 20 minutes. We made it clear that this was optional and would not affect the study's compensation. Three participants chose to use this grace period to finish the task they were working on.

At the end of the session, we asked the participants to answer five questions about their experience with the documents. In total, each session lasted approximately one hour. Participants did not need to do anything for the study before or after their session.

[3] Participants provided informed consent before the session. This study was approved by our institutional Ethical Review Board.

Table 1: Programming Tasks Used During the Study

#	Summary	Challenging Element
1	Sum all values in a column	Get familiar with the JDBC API
2	Print the content of a table	Use `wasNull()` to differentiate `0` and `NULL` values
3	Insert a new row in a table	Let the database generate the primary key
4	Insert untrustworthy string values	Prevent SQL injection with `PreparedStatement`
5	Create a new table	Create an auto-incrementing column
6	Insert rows as a single transaction	Handle transactions and rollbacks
7	Connect to a new database	Understand the syntax of the connection URL
8	Replace `NULL` values in a column	Understand the syntax of `UPDATE` statements

Participants could use their preferred integrated development environment (IDE) to solve the programming tasks locally on their computer. Alternatively, we also offered the option to work on a cloud development environment that was set up by the investigator. Two participants used this option. Throughout the session, we recorded the screen and audio of the participant using the videoconferencing software.

3.2 Programming Tasks

We designed eight tasks that required using Java's standard JDBC API to interact with a SQLite database (see Table 1). We chose this domain for the tasks as it requires understanding specialized concepts, such as the relational model of databases and transactions, and supported designing tasks of increasing complexity, from simple queries to table creation operations. Another consideration was to select a domain that participants would likely have some familiarity with to be able to solve a few of the tasks, but for which participants should not be able to solve all tasks easily without consulting the documentation. Most participants (ten out of thirteen) had prior experience with SQL, as our institution offers an undergraduate course on databases, but not with SQLite or JDBC specifically. We did not advertise the domain of the tasks during the recruitment phase, or used it as a selection criterion, to avoid influencing participants to study these technologies prior to their session.

All tasks were presented in the same Java file, and required only Java code. After cloning a Git repository, and potentially fixing configuration issues with the investigator, the participants only had to complete the body of one method per task. Peripheral issues, such as file paths, error handling, and code quality, were outside the scope of the tasks. The instructions of each task were presented as code comments in the empty method to complete.

Participants attempted the tasks in the order they were presented and had to successfully complete a task before moving on to the next one. The investigator instructed participants that they should not try to finish as fast as possible and that we did not expect them to finish all eight tasks during the session. Rather, the tasks served as a context to trigger information searches.

To verify whether they completed a task, participants ran the `main` method of the Java program. They could run the program as often as desired to test partial solutions, verify a solution, and debug issues. The start of the `main` method reset the database, so participants did not have to handle unintentional consequences of successive runs.

Table 2: Topic of the Six Documents Provided to Study Participants

Name	**Topic** (visible to participants)	**Tasks** (hidden)
Connection	Connect to the SQL database from a Java program	7
Create-Table	Create SQL tables and specify constraints on the columns	5
Read-Values	Read the content of a database	1, 2
Write-Values	Append rows to a table of the database	3, 8
Safe-SQL	Avoid SQL injection attacks when handling user-provided values	4
Transactions	Execute several SQL statements within a transaction	6

3.3 Documents Provided

Participants were only allowed to use the documents that we provided, the API reference documentation from Oracle's website, and the information available from their IDE.

We authored six documents for the study. Their content was based on popular tutorials, including Oracle's official "JDBC Basics" tutorial (Oracle 2022). We reused the sample application scenarios from the tutorials, but adapted the content to ensure that the documents covered all the information needed to solve the tasks. As it was common in tutorials to address several variations of SQL, we included information for four popular databases: SQLite, the database used in the tasks, in addition to MySQL, PostgreSQL, and Oracle Database. The documents did not emphasize SQLite-specific content over the rest, so participants had to ignore irrelevant content. However, only later tasks required vendor-specific adaptations.

Each document described one type of operation to perform with the JDBC API, as shown in Table 2. Participants were free to consult any of the documents at any time, but all the information for each task was contained in a single document. There was no indication of which document was relevant for a task.

All documents contained duplicated content in both Casdoc and a control format, as shown in Figure 2. Apart from their title and a one-line description, each document started with a complete code example (marked with an "A" in Figure 2). The code example contained interactive Casdoc annotations (marked with a "B", hidden when participants open a new document). Casdoc's search bar, which allowed participants to search within hidden annotations, was present in the top right corner. The content of all annotations was replicated below the code example in a familiar, non-interactive format (marked with a "C"). We organized the non-interactive content to maximize its readability. Instead of presenting all annotations as a series of separate fragments, we grouped related annotations into cohesive paragraphs and added headers to separate the main topics described in each document. The resulting layout of information is similar to tutorial documents found online. Henceforth, we refer to the code annotations and search bar (B) as the *interactive format*, and to the non-interactive text (C) as the *expanded format*, as it expands the content of all annotations in a single (scrolling) view. As the code example (A) is an essential component of both formats, it is not a *distinguishing* feature of either format in this study. Therefore, we did not consider it as part of either format in the analysis.

At the start of each session, the investigator told participants that both the interactive and expanded formats had exactly the same content, apart from headers and annotation titles. The only exception was for API reference documentation.

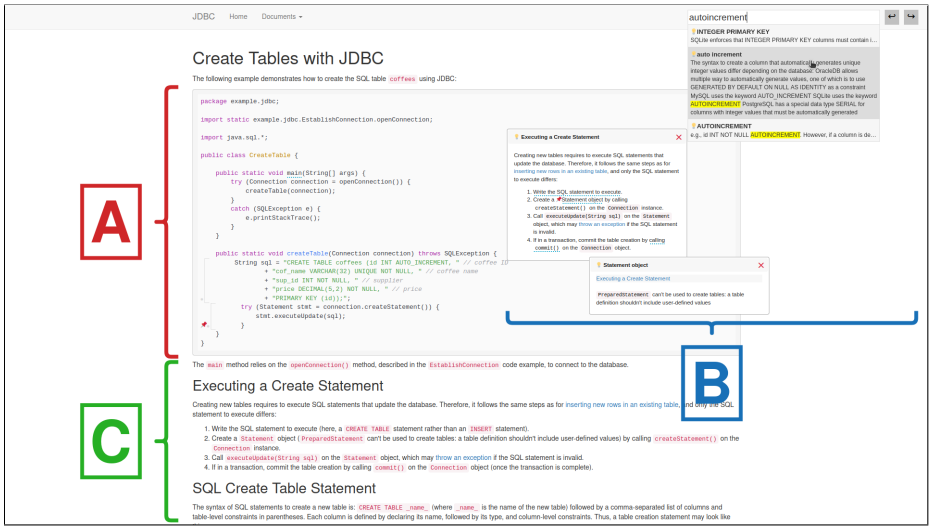

Fig. 2: Format of the documents provided to participants. Each document started with a code example (A), which included Casdoc annotations and the Casdoc search bar (B). The code example was followed by the expanded, non-interactive form of all annotations (C). When opening a new document, only parts A and C were visible.

Casdoc automatically generated annotations for API elements mentioned in the code (e.g., the class `String`). However, the expanded format did not contain that information to avoid documents that are too long, and because participants could access reference documentation from their IDE or from Oracle's website.

Combining the two formats in each document was a consequence of our study design with a single set of tasks. Given the small sample size and confounding factors such as the subjective preferences and backgrounds of participants, we opted for a within-subjects design in which each participant is exposed to both the interactive and expanded formats. In a conventional design, we would have asked all participants to work on two sets of tasks, using only one format for each set. However, this procedure would have required to create tasks that involved two distinct APIs to avoid carryover effects, introducing confounding variables due to the inevitable differences in the complexity of the tasks and quality of the documents. The conventional design would also have required long sessions, introducing the threat of participant fatigue during the second 40-minute set of tasks. Alternatively, shortening the time to 20 minutes per set of task would have reduced the amount of useful interaction with documentation: When piloting the study, participants required several minutes to get familiar with the documents and the API domain before progressing on the tasks. Thus, we employed a combined presentation that allowed participants to freely alternate between the interactive and the expanded formats.

Q1. Did any aspect of the documentation format helped or hampered your search for information within a document?
Q2. On a scale from 1 to 6, how challenging was it to find the information you were looking for in code annotations?
[scale shown below the question: 1 = extremely easy; 6 = extremely hard]
Q3. On a scale from 1 to 6, how challenging was it to find the information you were looking for in paragraphs below the code example?
[scale shown below the question: 1 = extremely easy; 6 = extremely hard]
Q4. Did the different formats affect your strategies for finding the information you wanted? If so, how?
Q5. Did the kind of information you looked for affect which format you used?

Fig. 3: Post-Study Questionnaire

3.4 Data Collection and Analysis

We collected data from the screen and audio recordings of participants during the programming tasks, in addition to the answers to the short questionnaire at the end of the sessions. The analysis relied on identifying qualitative patterns about the participants' search activities. Properties about the context of a search activity informed us about the *situations* in which one format is preferable to the other (RQ1). Navigation patterns revealed recurrent *sources of interference* between the formats and participants' actions (RQ2).

Figure 3 shows the five questions in the questionnaire. The role of this questionnaire was to prompt participants to reflect on their information search strategies and their interaction with the formats. We used the participants' responses to elicit initial patterns to answer our research questions, rather than to provide definitive conclusions. Therefore, we allowed participants to diverge from the questions when they described relevant search strategies or rationales. For example, although Q2 and Q3 required only a short answer, most participants explained their rating, providing further information about the relative importance of different aspects of the two formats. The first three questions, which relate to the ease of using each format, aimed at generating insights about navigation obstacles (RQ2). The last two questions, which relate to the interaction between the search activity and the format used, aimed at eliciting insights about the choice of format (RQ1). Nevertheless, we considered the responses to all questions when answering both research questions.

We started by summarizing the elements of the questionnaire responses that related to the document formats. We then operationalized each of these initial findings and investigated their occurrence during the programming tasks. The results consist of validated situations and navigation behavior that affect the ability of the interactive or expanded format to satisfy an information need. They contribute to the knowledge of documentation format design for programming knowledge by providing a set of hypotheses that link features of a format with a reader's interaction behavior.

The analysis required three phases of data annotation to transcribe the video recordings.

First phase: Session events. The first phase took place concurrently with the analysis of the questionnaire answers. We transcribed each session as a series of events related to interactions with the documents or to the tasks. Table 3 shows an excerpt of this

Table 3: Sample Session Events Transcribed During the First Analysis Phase

Time	Task	Window	Search	Doc. Component	Intervention
...					
35:10	Start 3				
36:00		Write-Values	Start	Text	
36:16				Code	
36:37				Popover “NULL”	
36:39				Dialog “NULL”	
37:16			Found		
37:27					Question
37:33					End
...					
38:14	Coding	IDE			
39:58		Write-Values	Quick	Code	
40:02				Dialog “NULL”	
40:04		IDE			
...					

* The online appendix describes in detail the meaning of the column values.

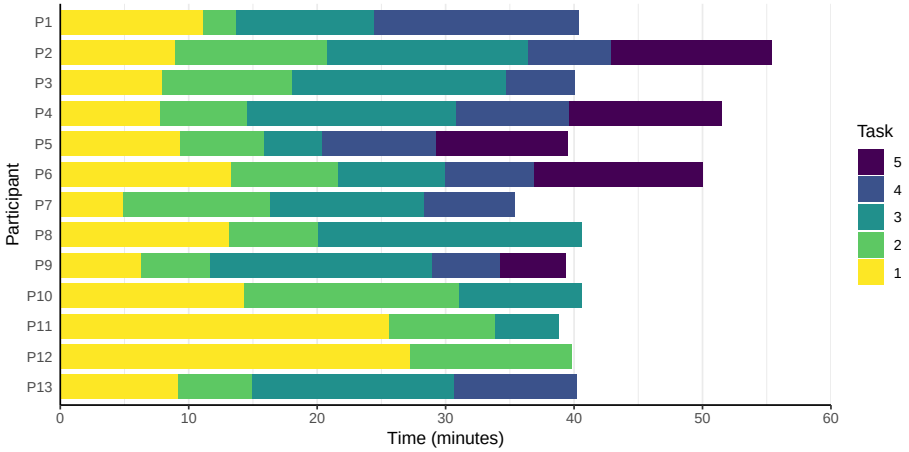

Fig. 4: Time spent per task for each participant. The chart excludes the time spent reading general instructions about the tasks context, which explains why not all bars reach a total of 40 minutes.

transcription. Each event has a timestamp, and belongs to one of five categories: *Task* events relate to the progress on tasks; *Window* events indicate which document the participant was looking at; *Search* events denote observable milestones of searches (e.g., starting a new search); *Doc. Component* events indicate which part of each document the participant interacted with; and finally *Intervention* events denote interactions between the participant and the investigator, such as the participant asking about the version of SQLite used. This phase provided a quantitative overview of the sessions, presented in Figure 4.

The bar chart excludes the time participants spent reading general instructions that applied to all tasks, e.g., instructing them to write all code in the same Java

file. This exclusion is the reason why not all bars reach a total of 40 minutes. P7 lost more time than others at the start because they wrongly assumed that they had to read all the documentation before starting the tasks. P11 and P12, who progressed the slowest, were also the least experienced, both having only one year of prior Java experience and no prior exposure to SQL. All other participants had at least two years of Java experience and all but one of them (P8) had prior exposure to SQL.

Second phase: Search fragments. For each participant, we split the study sessions into *search fragments*, which form our units of analysis. A search fragment consists of a period where the participant consults the documentation to search for some target information without making progress towards their solution. The fragment starts when a participant moves from their IDE to the online documentation, either opening a new document or revisiting an already opened one. The fragment ends when the participant shows that they are no longer looking for additional information, e.g., by stating so aloud or by moving back to the IDE. Two search fragments must be separated by time spent in the IDE working on the solution. The information sought during a search fragment may change, for example, if a participant finds some unexpected information that makes them reconsider their strategy.

In some cases, especially when trying to resolve issues, a participant would rapidly alternate between looking into the documentation and trying various ways to progress. We considered such situations as a single search fragment, as the participant stops and resumes the same search activity, until they either find a solution or abandon trying to solve the issue (e.g., by changing their approach to the task). In both cases, we interpreted this resolution as progress in the task.

In contrast, participants searching for information only within their IDE produced search fragments that are outside the scope of this study, and excluded from the results, because they do not involve documentation found *online*. Within-IDE search fragments differ from those that use online documents as they do not involve a context switch (Meyer et al. 2014). In practice, these fragments are also difficult to operationalize and delineate, as they are mostly short, frequent, and interleaved with writing code. We also excluded short interactions with the documents, such as locating known information, as they do not require the participant to *search* for information.

Finally, we excluded search fragments for which the target information was not part of the provided documents. For example, some participants looked for reminders about details of the Java language, such as the syntax of `for` loops or common operations on `String` objects. As we did not want participants to use resources other than the curated documents, the investigator provided the missing information or indicated that it was irrelevant to the task. In these situations, the investigator was careful not to give hints about concepts related to SQL or the JDBC API explained in the documents.

In total, we identified 122 search fragments, summarized in Table 4. For each search fragment, we extracted properties about the context in which the search took place and properties related to the navigation within the documents. Tables 5 and 6 show the context and navigation properties, respectively, of the search fragments of P13 as an example of the data generated during the second phase.

The context of a search includes the related task, the documents consulted, and the start and end timestamps relative to the video recording, from which we computed the duration of the search fragment. The context also includes a categorization of the

Table 4: Summary of the Search Activity Per Participant

Part.	**# Frag.**	**Avg. Time** (seconds)	**Total Time** (minutes)	**# Frag. that Use Each Format** Code only	Interactive	Expanded
P1	10	50	8.3	3	3	5
P2	13	90	19.5	4	7	6
P3	9	69	10.4	2	2	7
P4	14	78	18.2	1	8	9
P5	11	89	16.3	1	10	6
P6	8	42	5.6	0	1	8
P7	7	123	14.4	1	5	2
P8	10	72	12.0	2	7	2
P9	13	71	15.4	3	9	5
P10	10	97	16.2	1	6	8
P11	5	259	21.6	0	5	4
P12	4	318	21.2	0	2	3
P13	8	61	8.1	1	6	5
Total	122	92	187.1	19	71	70

Table 5: Excerpt from the Second Analysis Phase: Context of the Search Fragments Performed by Participant P13

ID	**Start**	**End**	**Task**	**Intention**	**Document(s)**
115	21:16	21:48	1	Initial solution	Read-Values
116	32:55	34:03	2	Specific aspect	Read-Values, Javadoc ResultSet
117	35:59	37:16	3	Specific aspect	Write-Values, Create-Table
118	41:21	42:11	3	Specific aspect	Write-Values
119	42:41	43:44	3	Specific aspect	Write-Values
120	45:54	46:20	3	Specific aspect	Read-Values
121	48:57	49:49	3	Fix issue	Write-Values
122	53:26	55:27	4	Get familiar	Safe-SQL

* The online appendix describes in detail the meaning of the column values.

intention of the search. We started by labeling each search fragment using open codes, based on the nature of the information consulted, the participant's behavior when finding information, and the current progress on the task. We constantly compared new codes with the previous ones to ensure consistency. We refined the categories as we progressed, until we reached a stable set of four categories, such as looking for an initial solution template or for a way to fix an issue.

The navigation properties include the list of document *components* that the participant interacted with and the *sequence* in which each component was used. We identified the components based on mouse interaction, the information participants read aloud, and other visible signs. For example, using the participants' video recording, we could distinguish whether they were looking at the top code example or at the expanded text when both were visible. Some participants also read aloud fragments of the documents. The navigation properties also include the type of information sought when using either format (*Information Type (Interactive/Expanded)*). We categorized information types using the same procedure as for the search intention: starting with open codes and refining them as we progressed until we reached a stable set. We inferred the information sought based on the information previously found,

Table 6: Excerpt from the Second Analysis Phase: Navigation Properties of the Search Fragments Performed by Participant P13

ID	Components	Sequence	Information Type	
			Interactive	Expanded
115	code	*new* → *code*		
116	code, text, popover	*known* → *code* → *text* → *code* → *popover* → *link ; javadoc* → *text*	link	content overview, element detail
117	code, text, popover, dialog	*new* → *code* → *text* → *code* → *popover* → *dialog* → *link ; new* → *code* → *text* → *code* → *back ; known* → *dialog*	continue "how to"	how to, content overview
118	dialog, popover, text	*known* → *dialog [*↔ *popover]* → *text*	element detail	confirm "content overview"
119	code, text, popover, dialog, unint. popover	*known* → *code* → *text* → *code [*↔ *popover]* → *dialog* → *link ; javadoc* → *text*	continue "how to"	content overview, how to
120	code, popover	*known* → *code* ↔ *popover*	element detail	
121	code, popover, dialog	*known* → *code* ↔ *popover* → *dialog [*↔ *popover]* ↔ *ide*	error cause	
122	code, text, unint. popover	*new* → *code* → *text* → *code [*↔ *ide] [*↔ *text]*		concept

* The online appendix describes in detail the meaning of the column values.

the information needed to progress on the task, and the participant's reaction when finding new information.

The same investigator who conducted the sessions extracted the properties for the search fragments, as this task required a close familiarity with the sessions. During the sessions, the investigator made sure to understand the search activities (e.g., their intention and information sought), prompting the participant for clarifications when necessary. We categorized the search intentions and information types inductively from the participants' actions and comments. The narrow context of the programming tasks considerably mitigated the difficulty of extracting these properties, and we kept the categories broad to reflect only what we could reliably observe.

Third phase: Navigation patterns. The third phase consisted of synthesizing recurring navigation patterns (presented in Section 4) and marking, for each search fragment, whether the fragment contains an occurrence of the pattern.

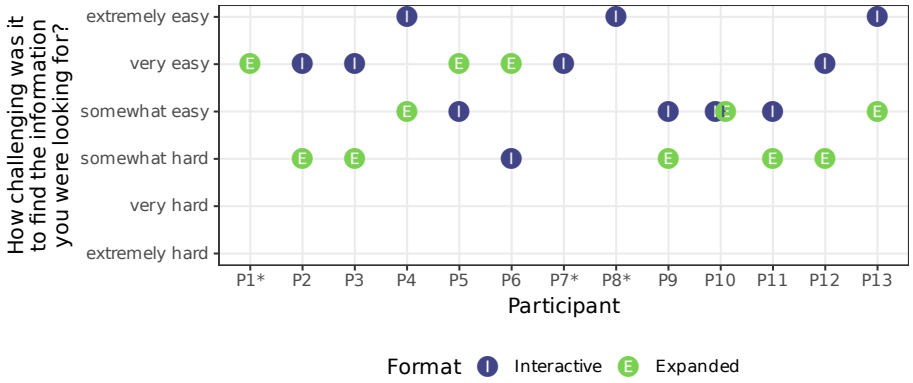

Fig. 5: Participants' Ratings of the Interactive and Expanded Formats. Three participants (marked by *) rated only one format.

4 Results

Figure 5 shows the subjective ratings of the interactive and expanded formats. Seven participants indicated that the information was easier to find in the interactive format than in the expanded one, two indicated the opposite, and one indicated that information was equally easy to find in both formats. Three participants preferred not to rate one of the formats because they only remembered using annotations (P7 and P8) or the expanded text (P1). Although these ratings favor the interactive format, they must be interpreted carefully. Participants in user studies may be biased to view experimental tools favorably (Dell et al. 2012). However, all participants found at least one format (extremely, very, or somewhat) easy to navigate. These results indicate that the documents did not have significant readability issues that would invalidate this study.

Despite their preferences, all participants used both formats during their session to find information, including those who rated only one format. In answer to RQ1, the analysis of the programming sessions and of the post-study questionnaire elicited **factors related to four aspects of an information search that may affect the choice of a format** (Section 4.1). In answer to RQ2, the study also revealed **how Casdoc supported or interfered with three recurrent groups of navigation patterns, which can affect a programmer's preference for a format** (Section 4.2).

Overall, the results show that the two formats can coexist in the same document and provide complementary benefits. Despite the duplication of all the content, participants appreciated having the possibility to switch between the two formats according to their evolving needs. We discuss the implications of our results in Section 4.3.

4.1 Choice of Documentation Format (RQ1)

Participants related their use of the interactive and expanded formats to the need to find different types of information. For example, P5 expressed *"The general 'how to do a thing' was definitely more for the code highlights. [...] If I wanted to fix a bug or something, I looked through the paragraphs."* Similarly, P10 mentioned *"When I just want to search a keyword, I prefer just using the [popover]."* We investigated this behavior at two levels. First, for each search fragment, we attributed one of four broad categories of **search intention**. The intention captured the purpose of the sought information in relation to the task. Second, we identified the **type of information** that participants looked for every time they switched to a different format during an information search fragment. If the information type changed as the search progressed, we only considered the information type that triggered the initial usage of a format.

Another recurring strategy from participants was to use the code and the interactive format first when looking for information, then move to the expanded format if they could not find what they were looking for. For example, P9 said *"I would first look at the code annotations. Then, if there's any detail that's missing, I don't really bother to go over all those levels of popovers to look for it. I would just go through the paragraphs."* P3 expressed more succinctly *"I always go to the code annotations first, and then try reading the paragraphs."* These responses hinted at a consistent **sequence of formats** when participants did not know in advance where the information would be located (e.g., from a prior search fragment).

Finally, multiple participants indicated that they perceived the expanded format as more **reliable and complete** than the interactive format, despite being shown that both formats contain the same content. P6 stated this bias clearly: *"I don't know if that's true, but I assumed that there's going to be more information in the paragraphs."* P10 also said *"I thought the documentation below has more thorough explanation."*

Search intention. We distinguished four broad categories of search intentions: *getting familiar* with the API or domain concepts, looking for an *initial solution* in the document that can be adapted, finding information about a *specific aspect* of the task, or *fixing an issue*. Search fragments categorized as *getting familiar* or *initial solution* typically occurred at the beginning of a task, whereas *specific aspect* and *fixing an issue* typically occurred after the participant had made some progress on the task. The category *getting familiar* applied to search fragments where the participant gathered information about many elements related to a broad context. For example, some participants sought to learn about JDBC's `Driver` and `Connection` objects, including the abstract concepts they represent and their methods, before looking for information about the objective of the first task (i.e., reading values in a table). In contrast, the categories *specific aspect* and *fixing an issue* applied to search fragments where the participant sought to answer a narrow question that would help them accomplish a part of the task or fix a problem in their current solution. For example, participants looked for how to distinguish between `0` and `NULL` values when using the `getInt(String)` method to read a table (*specific aspect*), or for the cause of an exception thrown by the program (*fixing an issue*).

Table 7 shows the number of search fragments that used either format for each intention. When looking for a specific aspect of the task or a solution to an issue,

Table 7: Search Intentions and Formats Used During Search Fragments

Format	**Search Intention**			
	Get familiar	Initial solution	Specific aspect	Fix issue
Code only	0	13	3	3
Interactive	1	12	16	4
Expanded	3	7	17	5
Both	3	6	25	4
Total	7	38	61	16

Table 8: Usage of Each Format to Look for Information of Each Type

Format	**Information Type**					
	how to	error cause	element detail	example implementation	SQL syntax	concept
Interactive	10	2	27	13	13	3
Expanded	18	7	6	2	17	5
Total	28	9	33	15	30	8

participants used each format almost as often. However, when looking for an initial solution to the task, participants used interactive annotations more often than the expanded text to support their initial understanding of the code example. In contrast, although it did not happen frequently, reading the expanded format was preferred when a participant wanted to get familiar with the domain of a task without a specific query in mind.

Type of Information. Participants often looked for the same recurring information as they proceeded through the tasks. We identified six categories of information needs specific to our experiment based on how participants expressed what they looked for: information about *how to* perform an operation, *error causes* and potential solutions, API *element details*, clarifications about the *code example implementation*, details about the *SQL syntax*, and *conceptual* information about the task or API domain. When a participant used both formats to search for the same information, only the first format is considered, as it was the participant's initial instinct.

Some categories, such as *element details* and *code example implementation*, capture needs that are closed-ended: participants have a prior idea of the type and extent of information required to satisfy their need. For example, participants searched for the behavior of the `getInt(String)` method when a table contains a `NULL` value (*element detail*). Satisfying this need requires a short, factual answer (i.e., the method returns `0`). In contrast, other categories such as *how to* and *error cause* capture open-ended needs: participants did not have a prior notion of the solution to their need. These needs typically required them to synthesize information from multiple locations. For example, learning *how to* distinguish `0` from `NULL` values when the method `getInt(String)` returns `0` requires to look for other relevant methods, usage examples, and API directives.

Table 8 shows the usage frequency of both formats for each information type. Participants predominantly used the interactive format for details about an API element and clarifications of the main code example's implementation. In contrast,

they sought "how to" instructions and error causes more often in the expanded format. Information about SQL syntax and concepts were more balanced between both formats, with a preference for the expanded format.

Sequence of formats. Many participants mentioned looking for information first in the interactive annotations, and then in the expanded text if it took too long to find the information. However, when identifying instances where a participant changed format when looking for the same information, we observed that they went from the interactive format to the expanded format 15 times, and the other way around 14 times. This suggests that, regardless of the initial format, when participants could not find some information, changing format was a viable strategy to continue the search.

However, a closer look at the instances revealed that participants mostly (11 out of 14 times) relied on the search feature of Casdoc after failing to find information in the expanded text. This suggests a more nuanced sequence of formats: first interacting with the annotations from the code example, then looking into the expanded text, and finally using a text search feature to locate hard-to-find information.

Perceived reliability and completeness. Despite an explicit mention that both formats had the exact same content at the start of each session, participants showed a bias to trust more the expanded format. We did not expect this bias, and thus did not measure this subjective aspect of the format, e.g., with questions in the post-study questionnaire.

We nevertheless gathered evidence of this bias by identifying instances where participants used one format to confirm information already found. Five search fragments, from four distinct participants, showed such behavior, all using the expanded format. In these instances, participants expressed the desire to scan the text in case they missed relevant information. For example, one participant mentioned that *"Somehow I want to see it [the information] in the full [expanded] documentation. [...] I thought the documentation below has more thorough explanations"* (P10).

4.2 Support of Navigation Actions (RQ2)

Participants referred to three recurrent groups of navigation actions during the study. One of the most common actions was to first **look for code examples** when reading a new document. For example, P1 stated *"I prefer to look directly at some code which is simple but really good. [...] Even if the code is at the end of a document, I always try to look at it."* This sentiment was echoed by P2, among others: *"I always default to code first."* Even after reading the main code example, some participants looked through the document for further examples, such as P12, who indicated while scanning the expanded text *"I am looking for examples, because that's how I learn best."*

Another common action was to **scan a document's content** without a single specific information need. Participants performed this action to locate where a given topic was discussed, to assess the extent of the coverage of a topic, or to get an overview of the set of topics discussed in a document. For example, P6 discussed having prior strategies for reading expanded text efficiently, and said *"I guess my strategy is always to look for the keyword and then, once I see the keyword, I will*

read stuff around it." P9 and P12 also mentioned their appreciation for features that help scan documents by stating, respectively, *"One thing that I found really helpful was the subtitles."* and *"I'm also someone who really enjoys indentation in things, to make it easier to scan through."*

Finally, participants pointed out limitations related to popovers appearing unintentionally when they **read with the mouse**. P7 mentioned this issue directly: *"There were so many pop-ups. I've a tendency to move my mouse while I read the code. [...] That kind of interrupts the reading flow."* Similarly, popovers interfered with participants trying to copy and paste code, as P6 said: *"When I try to do copy-pasting, the pop-up window [...] actually interferes with the actual copy-pasting of the code."*

Looking for code examples. We measured the reliance on code examples by identifying when, if at all, participants used the main code example during search fragments. In 19 search fragments (16%), participants used only the code example. In 57 fragments, (47%), the code was the first element participants looked at, and in 16 fragments (13%), they looked at other components of the documents first. In the remaining 30 fragments (25%), participants did not use the code example. This prominent use of the code example is consistent with the participants' comments. Although it was not a distinguishing feature of the two formats in this study, the focus on a complete code example was one of the key design properties of Casdoc (Nassif and Robillard 2023). Our results confirm the validity of this design aspect for documentation.

Scanning a document's content. Casdoc does not support well scanning the content of a document, as it cannot provide an overview of all annotations at once. We measured how often participants did scan the expanded format to assess the impact of this limitation of the interactive format. We observed this action in 30 search fragments, performed by all but one participant. This represents 25% of all search fragments, or 43% of the fragments where the expanded format was used. The popularity of this action suggests that it should be a consideration for documentation designers, especially for interactive formats that do not show all of their content at once.

Reading with the mouse. Although many participants appreciated the swift interaction mechanism to reveal and hide popovers, some noted their distracting nature when performing other mouse actions. To measure this negative impact, we measured how often popovers opened by unrelated mouse movements distracted participants. A popover was *distracting* when it interfered with a participant trying to select some code to copy, when it hid the code that the participant was trying to read, or when it triggered an abrupt mouse motion to leave the anchor. We found instances of distracting popovers in 36 search fragments (30%), affecting all but one participant. These unintentional popovers did not count as a usage of the interactive format in the other metrics, e.g., when comparing how often participants used each format. This result highlights a trade-off when designing the interaction mechanism of a format: simplifying the actions to access content can generate accidental events.

4.3 Discussion

Our observations revealed noteworthy aspects of information searching strategies that have implications on documentation design, beyond the answers to our two research questions.

The results generally show that the interactive format allowed participant to get concise information to answer specific queries. However, queries that required more elaborate answers were better answered in the expanded format. The expanded format also made it easier for participants to scan information related to their original query. P10 gave the syntax of SQL conditions as an example: *"Sometimes I want to see other content related to 'WHERE' as well, [...] to see the overall picture."* Reading this related content typically required participants to spend more time than with the interactive format, but it could help them feel more confident that they correctly understood the information they found.

The personality and prior navigation behavior of a reader may also affect their preference for a format. How much a reader uses their mouse while reading may affect their opinion on whether the usefulness of pointer-triggered interactions to reveal contextual information outweigh their distracting potential. Some readers may also be more enthusiastic about adopting new navigation techniques for the interactive format, whereas others may be reticent to abandon the more familiar expanded format. Thus, the design of documentation should not only accommodate differences in the search contexts, but also in personal preferences.

An overarching theme among the results is that the combination of the interactive and expanded formats accommodated a variety of navigation techniques. Each format mitigated some limitations of the other, which allowed participants to use the most appropriate format depending on their needs. Participants noted this synergy: *"If I wanted very quick information, I would just look at the highlights that were directly in the code. [...] If I got stuck on something, or I needed a bit more information, then I went into the paragraphs to look for it. [...] The two levels were quite nice, with respect to how much information I needed"* (P5). Even the redundancy of the content, which is typically regarded as a negative aspect of documentation, had some advantages: *"It was kind of reassuring to know that there are different strategies for finding what I wanted. [...] It [duplicated content] made me more comfortable looking anywhere"* (P12). We did not expect such positive feedback on the dual nature of the documents' presentation, which we created to avoid long programming sessions. Future work could fully integrate the two formats, e.g., by allowing readers to jump between annotations and their expanded representation, to further improve documentation design.

In summary, the observations we collected during the study have the following implications for the design of interactive features in the format of software documentation. We indicate in parentheses the paragraphs from the previous sections that elaborate on each implication.

1. Low-effort interactive features (e.g., mouse hover) can contribute to a swift user interface, but they can also distract readers if they are triggered by unintentional actions. (Section 4.2: *Reading with the mouse*)
2. Readers must not only find the information they seek, but also gain confidence that they correctly understood that information. (Section 4.1: *Perceived reliability and completeness*)

3. Readers can change their search strategy (e.g., by changing format) when it takes too much time to find specific information, but additional time and effort can also increase the readers' confidence in the information they find. (Section 4.1: *Sequence of formats* and *Perceived reliability and completeness*)
4. Hiding a portion of a document's content can emphasize the most important elements, such as the code examples, but it makes it harder for readers to get an overview of the topics that the document addresses. (Section 4.2: *Scanning a document's content*)
5. Even within the context of a specific task, software documentation must support a variety of search strategies, influenced both by personal preferences of the readers and the context of the search, e.g., whether the target information is closed- or open-ended. (Section 4.1: *Search intention* and *Type of information*)
6. Documents can accommodate various information search strategies by using multiple complementary formats, even at the cost of duplicating content. (Section 4.1: *Sequence of formats*)

These implications complement prior work on information searching behavior. Readers desire low-effort and intuitive features to interact with documents (Tashman and Edwards 2011a). According to information foraging theory, minimizing the effort to navigate within a document is important given the abundance of documentation resources (Pirolli and Card 1999). Otherwise, readers are likely to search for other documents. However, Implication 3 suggests, as a possible intermediate step, that readers may change their search strategy before looking for other resources if the document allows it. Furthermore, Implication 1 highlights a trade-off between effort and intentionality in the design of interaction features. Designers should balance this trade-off based on the targeted usage context of the document or the feature (see, for example, the design discussion of Hinckley et al. 2012). Similarly, designers must balance the trade-off between providing readers the flexibility to move information fragments and ensuring that the context of these fragments is not lost (Tashman and Edwards 2011a). This trade-off adds to the tension expressed in Implication 4 in the design of modifiable presentation formats. Supporting multiple modes of interaction is also a recurring theme in prior studies, as readers have varying reading preferences and objectives when interacting with documents (Meng et al. 2018; Nam et al. 2024). Implication 5 highlights the importance of alternative interaction features not only for different readers, but also for the same reader performing searches in different contexts. Implication 6 suggests that (possibly conflicting) interaction modes can be constrained to different parts of a document.

Looking forward, our study may help refine the design of the new wave of documentation tools based on large language models (LLMs). Although research on LLM-based tools often focus on their abilities to *generate* information, researchers have started to also explore options for their user interfaces (Masson et al. 2024). Our results can contribute to this effort. For example, when prompted for a code example, a conversational agent could add explanations using the Casdoc format instead of (or in addition to) placing them below the code example. Our results can also help adjust subtle aspects of LLM-based tools. For example, the observation about the recurrent sequence of formats used by participants can be used to adapt the tone of a conversational agent and the amount of information in its responses: the agent could provide concise factual answers to its user's initial query, and gradually include more peripheral information and use a pedagogical tone for successive queries about

the same topic. Therefore, although our study employed Casdoc as the interactive format, the results should provide insights into various trade-offs and considerations for the design of other documentation interfaces.

4.4 Study Trade-Offs and Threats to Validity

Designing a study requires to balance multiple objectives and constraints, leading to trade-offs that impact the study's outcomes (Robillard et al. 2024). We chose to conduct a user study in a controlled environment to limit the number of factors that can influence the participants' behavior when using the documents. This uniform context across all participants, along with the presence of one investigator during the sessions, allowed us to better understand recurring patterns and search strategies. This study complements our initial field study of Casdoc (Nassif et al. 2022; Nassif and Robillard 2024b).

Forcing participants to use only the provided documents meant that they could not use familiar resources such as their favorite search engine or Stack Overflow. A small and predefined documentation set also impacts the aspect of information search related to navigating *between* documents. For example, participants had to be more persistent in searching within a single document than they would likely be in a normal setting. An alternative decision would be to provide the study documents, but allow participants to also look for other documents. This option would lessen the burden of authoring the study documents when preparing the study, as it is less critical that they contain information about all possible questions participants may have. However, the adoption cost of a new format may incite some participants to only use resources that they know, in which case the sessions would have to be discarded. Thus, we restricted documents to a closed set to encourage participants to try the interactive format during the sessions.

The study required participants to discover a new documentation format as part of the sessions. Participants were used to searching within the expanded format, and had developed techniques to do so more efficiently. P6 expressed this bias directly: *"Reading the paragraphs is more intuitive, because I'm already used to reading that kind of documentation."* In particular, many participants (P4, P6, P7, and P13) indicated that redirecting the keyboard shortcut Ctrl+F (or Cmd+F) to Casdoc's search bar, instead of the browser's native search tool, negatively affected their experience.[4] In contrast, participants had to learn about Casdoc's features and develop an intuition about how and when to use them. Participants reacted differently to this change. Some were more enthusiastic and tried more advanced features of Casdoc, whereas others preferred to rely mostly on their existing strategies. It is possible that programmers who have used Casdoc for more than the study's 40 minutes would elaborate new navigation approaches that the findings do not capture.

A threat to the validity of our observations is that the investigators designed both the tasks and the documents. We ensured that both formats in each document contained exactly the same content, which was based on existing online tutorials. It is possible, however, that other design aspects of the documentation, such as the order of paragraphs and sections in the static text, would affect their navigability.

[4] The browser's search tool remained available without the keyboard shortcut, but participants had to open it from the browser menu.

Regarding the tasks, we designed them to represent common database operations, while also involving more challenging parts of the JDBC API so that participants would need to consult the documentation. To mitigate the potential of an investigator bias on the results, the analysis focused on qualitative similarities and differences between the formats, rather than on quantitative measures of properties (e.g., time) used as a proxy for document quality.

The selection of the participant sample also limits the generalizability of the results. A smaller sample size permits a more detailed analysis of the programming sessions within a practical time budget. This approach was consistent with the exploratory nature of the study, and is a first step to generate hypotheses that can be tested in future studies using statistical methods over a large sample. Using computer science students as the sampling frame also limits the generalizability to the target population. Although students are a part of the target population, they are not representative of the entire population. For example, experienced developers may have developed more proficient information search strategies and have more in-depth domain knowledge, which affect the information they would look for. Using a broad sampling frame, however, would increase the variability in the background, experience, and professional situation of the participants. In this case, a small sample size would have been insufficient to account for these confounding variables. Instead, we chose a well-known, but less representative, sampling frame that supports a better interpretation of our results.

5 Conclusion

The design of new interactive formats to present software documentation has the potential to improve how programmers learn to use unfamiliar APIs. We conducted a user study to understand the impact that an interactive format, Casdoc, has on programmers' navigation patterns. During the study, each participant worked on a series of tasks that involved using a Java API. We provided a set of documents that combined an interactive format with an expanded version of the same information, and recorded how participants used the documents through a 40-minute programming session. Participants also answered a short questionnaire at the end of their session to reflect on the documentation formats and their navigation strategies.

The analysis of the questionnaire answers and screen recordings revealed that the most appropriate format varies depending on the context and intention of the search activity. For example, although the interactive format helped participants find the information they sought more directly, the expanded format helped them feel more confident about the information they found. All participants used both the interactive and the expanded formats. We also observed how different patterns of navigation interfered with the formats. For example, mouse-based interactions can support fast and intuitive navigation actions, but they can also distract programmers who move their mouse while reading. Such interference can influence subjective preferences towards one format or another. Thus, a combination of multiple formats, even with duplicated content, can support a larger audience performing a broader range of searches than any single format.

Acknowledgements We thank the members of our research group for their help during the development and piloting phase of the study instrument. We also thank the study participants

for their contribution to the evaluation of documentation formats. Finally, we thank the anonymous reviewers for their constructive comments on our work.

Conflict of interest

The authors declare that they have no conflict of interest.

References

Aghajani E, Nagy C, Vega-Márquez OL, Linares-Vásquez M, Moreno L, Bavota G, Lanza M (2019) Software documentation issues unveiled. In: Proceedings of the IEEE/ACM 41st International Conference on Software Engineering, pp 1199–1210

Bäuerle A, Cabrera ÁA, Hohman F, Maher M, Koski D, Suau X, Barik T, Moritz D (2022) Symphony: Composing interactive interfaces for machine learning. In: Proceedings of the CHI Conference on Human Factors in Computing Systems, pp 210:1–14

Beyer S, Macho C, Di Penta M, Pinzger M (2020) What kind of questions do developers ask on Stack Overflow? A comparison of automated approaches to classify posts into question categories. Empirical Software Engineering 25(3):2258–2301

Bouraffa A, Maalej W (2020) Two decades of empirical research on developers' information needs: A preliminary analysis. In: Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops, pp 71–77

Brandt J, Guo PJ, Lewenstein J, Dontcheva M, Klemmer SR (2009) Two studies of opportunistic programming: Interleaving web foraging, learning, and writing code. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp 1589–1598

Chen N, Guimbretiere F, Sellen A (2012) Designing a multi-slate reading environment to support active reading activities. ACM Transactions on Computer-Human Interaction 19(3):18:1–35

Chi PY, Ahn S, Ren A, Dontcheva M, Li W, Hartmann B (2012) MixT: Automatic generation of step-by-step mixed media tutorials. In: Proceedings of the 25th annual ACM symposium on User interface software and technology, pp 93–102

Chimalakonda S, Venigalla ASM (2020) Software documentation and augmented reality: Love or arranged marriage? In: Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp 1529–1532

Dell N, Vaidyanathan V, Medhi I, Cutrell E, Thies W (2012) "yours is better!" participant response bias in HCI. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp 1321–1330

Duala-Ekoko E, Robillard MP (2012) Asking and answering questions about unfamiliar APIs: An exploratory study. In: Proceedings of the 34th IEEE/ACM International Conference on Software Engineering, pp 266–276

Feng KJK, Coppock MJ, McDonald DW (2023) How do UX practitioners communicate AI as a design material? artifacts, conceptions, and propositions. In: Proceedings of the ACM Designing Interactive Systems Conference, pp 2263–2280

Fittkau F, Krause A, Hasselbring W (2017) Software landscape and application visualization for system comprehension with ExploreViz. Information and Software Technology 87:258–277

Glassman EL, Zhang T, Hartmann B, Kim M (2018) Visualizing API usage examples at scale. In: Proceedings of the CHI Conference on Human Factors in Computing Systems, pp 580:1–12

Guo PJ (2013) Online Python tutor: Embeddable web-based program visualization for CS education. In: Proceedings of the 44th ACM technical symposium on Computer science education, pp 579–584

Head A, Appachu C, Hearst MA, Hartmann B (2015) Tutorons: Generating context-relevant, on-demand explanations and demonstrations of online code. In: Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing, pp 3–12

Head A, Glassman EL, Hartmann B, Hearst MA (2018) Interactive extraction of examples from existing code. In: Proceedings of the CHI Conference on Human Factors in Computing Systems, pp 85:1–12

Heer J, Card SK, Landay JA (2005) prefuse: A toolkit for interactive information visualization. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp 421–430

Higuchi K, Sano S, Igarashi T (2021) Interactive hyperparameter optimization with paintable timelines. In: Proceedings of the ACM Designing Interactive Systems Conference, pp 1518–1528

Hinckley K, Bi X, Pahud M, Buxton B (2012) Informal information gathering techniques for active reading. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp 1893–1896

ter Hoeve M, Sim R, Nouri E, Fourney A, de Rijke M, White RW (2020) Conversations with documents: An exploration of document-centered assistance. In: Proceedings of the Conference on Human Information Interaction and Retrieval, pp 43–52

Hornbæk K, Frøkjær E (2003) Reading patterns and usability in visualizations of electronic documents. ACM Transactions on Computer-Human Interaction 10(2):119–149

Horvath A, Liu MX, Hendriksen R, Shannon C, Paterson E, Jawad K, Macvean A, Myers BA (2022) Understanding how programmers can use annotations on documentation. In: Proceedings of the CHI Conference on Human Factors in Computing Systems, pp 69:1–16

Hu X, Li G, Xia X, Lo D, Jin Z (2020) Deep code comment generation with hybrid lexical and syntactical information. Empirical Software Engineering 25(3):2179–2217

Khandwala K, Guo PJ (2018) Codemotion: Expanding the design space of learner interactions with computer programming tutorial videos. In: Proceedings of the 5th Annual ACM Conference on Learning at Scale, pp 57:1–10

Kim DH, Hoque E, Kim J, Agrawala M (2018) Facilitating document reading by linking text and tables. In: Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology, pp 423–434

Ko AJ, Myers BA, Coblenz MJ, Aung HH (2006) An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. IEEE Transactions on Software Engineering 32(12):971–987

Liu J, Baltes S, Treude C, Lo D, Zhang Y, Xia X (2021a) Characterizing search activities on Stack Overflow. In: Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp 919–931

Liu M, Peng X, Marcus A, Xing S, Treude C, Zhao C (2021b) API-related developer information needs in Stack Overflow. IEEE Transactions on Software Engineering 48(11):4485–4500

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

MacLeod L, Bergen A, Storey MA (2017) Documenting and sharing software knowledge using screencasts. Empirical Software Engineering 22(3):1478–1507

Masson D, Malacria S, Lank E, Casiez G (2020) Chameleon: Bringing interactivity to static digital documents. In: Proceedings of the CHI Conference on Human Factors in Computing Systems, pp 432:1–13

Masson D, Malacria S, Casiez G, Vogel D (2023) Charagraph: Interactive generation of charts for realtime annotation of data-rich paragraphs. In: Proceedings of the CHI Conference on Human Factors in Computing Systems, pp 146:1–18

Masson D, Malacria S, Casiez G, Vogel D (2024) DirectGPT: A direct manipulation interface to interact with large language models. In: Proceedings of the CHI Conference on Human Factors in Computing Systems, pp 975:1–16

van der Meij H, van der Meij J (2014) A comparison of paper-based and video tutorials for software learning. Computers & Education 78:150–159

Mellis DA, Zhang B, Leung A, Hartmann B (2017) Machine learning for makers: Interactive sensor data classification based on augmented code examples. In: Proceedings of the Conference on Designing Interactive Systems, pp 1213–1225

Meng M, Steinhardt S, Schubert A (2018) Application programming interface documentation: What do software developers want? Journal of Technical Writing and Communication 48(3):295–330

Meyer AN, Fritz T, Murphy GC, Zimmermann T (2014) Software developers' perceptions of productivity. In: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp 19–29

Miller BN, Ranum DL (2012) Beyond PDF and ePub: Toward an interactive textbook. In: Proceedings of the 17th ACM annual conference on Innovation and technology in computer

science education, pp 150–155
Miniukovich A, De Angeli A, Sulpizio S, Venuti P (2017) Design guidelines for web readability. In: Proceedings of the Conference on Designing Interactive Systems, pp 285–296
Moslehi P, Rilling J, Adams B (2022) A user survey on the adoption of crowd-based software engineering instructional screencasts by the new generation of software developers. Journal of Systems and Software 185:111144:1–21
Nam D, Macvean A, Myers BA, Vasilescu B (2024) Understanding documentation use through log analysis: A case study of four cloud services. In: Proceedings of the CHI Conference on Human Factors in Computing Systems, pp 937:1–17
Nassif M, Robillard MP (2023) A field study of developer documentation format. In: Extended Abstracts of the ACM CHI Conference on Human Factors in Computing Systems, pp 7:1–7
Nassif M, Robillard MP (2024a) Appendix to "Evaluating interactive documentation for programmers". URL https://zenodo.org/doi/10.5281/zenodo.10637078
Nassif M, Robillard MP (2024b) Non-linear software documentation with interactive code examples. ACM Transactions on Software Engineering and Methodology pp 1–29, accepted for publication
Nassif M, Horlacher Z, Robillard MP (2022) Casdoc: Unobtrusive explanations in code examples. In: Proceedings of the 30th IEEE International Conference on Program Comprehension, pp 631–635
Oracle (2022) Lesson: JDBC basics. URL https://docs.oracle.com/javase/tutorial/jdbc/basics/index.html
Pirolli P, Card S (1999) Information foraging. Psychological Review 106(4):643–675
Ponzanelli L, Bavota G, Di Penta M, Oliveto R, Lanza M (2014) Prompter: A self-confident recommender system. In: Proceedings of the IEEE International Conference on Software Maintenance and Evolution, pp 577–580
Robillard MP, DeLine R (2011) A field study of API learning obstacles. Empirical Software Engineering 16(6):703–732
Robillard MP, Treude C (2020) Understanding Wikipedia as a resource for opportunistic learning of computing concepts. In: Proceedings of the 51st ACM Technical Symposium on Computer Science Education, pp 72–78
Robillard MP, Arya DM, Ernst NA, Guo JLC, Lamothe M, Nassif M, Novielli N, Serebrenik A, Steinmacher I, Stol KJ (2024) Communicating study design trade-offs in software engineering. ACM Transactions on Software Engineering and Methodology 33(5):112:1–10
Sillito J, Murphy GC, De Volder K (2008) Asking and answering questions during a programming change task. IEEE Transactions on Software Engineering 34(4):434–451
da Silva RFG, Roy CK, Rahman MM, Schneider KA, Paixão K, Dantas CEdC, de Almeida Maia M (2020) CROKAGE: effective solution recommendation for programming tasks by leveraging crowd knowledge. Empirical Software Engineering 25(6):4707–4758
Sun J, Xing Z, Chu R, Bai H, Wang J, Peng X (2019) Know-how in programming tasks: From textual tutorials to task-oriented knowledge graph. In: Proceedings of the IEEE International Conference on Software Maintenance and Evolution, pp 257–268
Suzuki R, Soares G, Glassman E, Head A, D'Antoni L, Hartmann B (2017) Exploring the design space of automatically synthesized hints for introductory programming assignments. In: Proceedings of the CHI Conference Extended Abstracts on Human Factors in Computing Systems, pp 2951–2958
Tashman CS, Edwards WK (2011a) Active reading and its discontents: The situations, problems and ideas of readers. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp 2927–2936
Tashman CS, Edwards WK (2011b) LiquidText: A flexible, multitouch environment to support active reading. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp 3285–3294
Todi K, Leiva LA, Buschek D, Tian P, Oulasvirta A (2021) Conversations with GUIs. In: Proceedings of the ACM Designing Interactive Systems Conference, pp 1447–1457
Tomova M, Hofmann M, Hütterer C, Mäder P (2024) Assessing the utility of text-to-SQL approaches for satisfying software developer information needs. Empirical Software Engineering 29(1):15:1–48
Treude C, Robillard MP, Dagenais B (2015) Extracting development tasks to navigate software documentation. IEEE Transactions on Software Engineering 41(6):565–581
Wang Y, Kim YS (2023) Making data-driven articles more accessible: An active preference learning approach to data fact personalization. In: Proceedings of the ACM Designing

Interactive Systems Conference, pp 1353–1366
Wu D, Jing XY, Zhang H, Feng Y, Chen H, Zhou Y, Xu B (2023) Retrieving API knowledge from tutorials and Stack Overflow based on natural language queries. ACM Transactions on Software Engineering and Methodology 32(5):109:1–36
Xiao T, Treude C, Hata H, Matsumoto K (2024) DevGPT: Studying developer-ChatGPT conversations. In: Proceedings of the 21st International Conference on Mining Software Repositories, pp 227–230
Yan L, Kim M, Hartmann B, Zhang T, Glassman EL (2022) Concept-annotated examples for library comparison. In: Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology, pp 65:1–16
Ye Z, Yuan X, Gaur S, Halfaker A, Forlizzi J, Zhu H (2021) Wikipedia ORES explorer: Visualizing trade-offs for designing applications with machine learning API. In: Proceedings of the ACM Designing Interactive Systems Conference, pp 1554–1565