Non-Linear Software Documentation with Interactive Code Examples

MATHIEU NASSIF and MARTIN P. ROBILLARD, McGill University, Canada

Documentation enables sharing knowledge between the developers of a technology and its users. Creating quality documents, however, is challenging: Documents must satisfy the needs of a large audience without being overwhelming for individuals. We address this challenge with a new document format, named Casdoc. Casdoc documents are interactive resources centered around code examples for programmers. Explanations of the code elements are presented as annotations that the readers reveal based on their needs. We evaluated Casdoc in a field study with over 300 participants who used 126 documents as part of a software design course. During the study, the majority of participants adopted Casdoc instead of a baseline format and used interactive annotations to reveal additional information about the code example. Although participants collectively viewed the majority of the documents' content, they individually revealed a minority of the annotations they saw. We gathered insights into five aspects of Casdoc that can be applied to other formats, and highlighted five lessons learned to improve navigability in online documents.

CCS Concepts: • **Human-centered computing → Field studies**; *Web-based interaction*; Interactive systems and tools; • **Software and its engineering → Documentation**; • **Social and professional topics →** *Computer science education.*

Additional Key Words and Phrases: field study, software documentation, documentation format, interactive documents, code examples

ACM Reference Format:
Mathieu Nassif and Martin P. Robillard. 2023. Non-Linear Software Documentation with Interactive Code Examples. 1, 1 (August 2023), 32 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

We present Casdoc, a novel technology for improving the presentation of online learning resources for programmers. Casdoc, which stands for Cascading documentation, presents the content of an HTML document as a set of interlinked, concise, and interactive annotations rooted in a code example. A transformation tool simplifies the authoring process for these documents by generating them from annotated code files.

Novel presentation approaches are needed to improve the way information seekers, such as programmers, use documentation. Documentation is a crucial asset to understand an unfamiliar software system [26, 61]. Yet, creating good documents requires a lot of effort and expertise. Software engineering researchers have proposed different techniques to generate content (e.g., [13, 41, 52]) or retrieve it from knowledge bases (e.g., [20, 57, 80]), which can alleviate some of this effort. However, documentation quality is multi-faceted [34]: it must not only contain enough information to address the concrete needs of its audience [18], but the information must also be readable, navigable, and understandable [3, 79]. These aspects, which relate to *how* the information is organized and presented, have not been studied as extensively. Prior work has proposed alternatives to navigate the content of documents, e.g., by presenting a knowledge graph of the document's content [15], a list of programming tasks explained in the document [73], or the combination of

Authors' address: Mathieu Nassif, mnassif@cs.mcgill.ca; Martin P. Robillard, robillard@acm.org, School of Computer Science, McGill University, Montréal, QC, Canada.

both [69]. Other approaches expand the content of one document by annotating code examples with content from other sources, e.g., GitHub [59] or API reference documentation [68]. However, these research efforts focused on the automated extraction of the underlying information structure (e.g., traceability links), instead of the readers' behavior when interacting with the new formats. As a result, current documentation formats can fail to emphasize the most useful fragments when too much content is available [82].

Casdoc is a solution to improve the navigability of content in code-oriented documents. In a Casdoc document, readers interact with code elements to reveal further explanations of those elements (see Figure 1 in Section 2). Information about elements that are irrelevant to a reader remains hidden to avoid unnecessary distractions. Casdoc relies on popovers and dialogs to achieve this objective. Hence, it recasts two graphical elements which are typically used for secondary navigation aid as the primary structure to organize the content of a document, now split into concise annotations. The resulting format is non-linear: it forces readers to select which part of the document to read next, as opposed to following an explicit order. Annotations are created by the document's author. Instead of writing prose that surrounds the code example, the author inserts explanations directly into working code files as code comments. This authoring process is similar to the generation of API reference documentation from header comments, but supports a different type of documentation (i.e., tutorials and other learning-oriented documents). The Casdoc transformation tool then converts the annotated code files into dynamic web documents.

Previous work has suggested approaches to improve the format of learning resources, as the transition from printed to digital documents created opportunities for new modes of interaction [31, 76]. Researchers proposed techniques to add interactive elements to existing static documents, such as data visualizations [7, 47, 48], custom annotations [33], and automatically generated links to external resources [6]. Other work suggested to make document visualization software more interactive with navigation features inspired by paper-based formats [66, 70]. Specifically for software documentation, researchers proposed to augment the interface of code search tools with explanations of how the code fragments were matched, to help programmers decide on the pertinence of the results [78]. Digital documents can also contain dynamic elements, such as runnable code examples [49, 77], explorable statistical analyses [22], or modifiable machine learning models [8]. All of these techniques, however, do not change the linear organization of information in existing documents, which does not always match the readers' navigation patterns [12, 32].

Casdoc challenges this traditional structure. We observed how programmers react to a non-linear format in a seven-month field study with 326 participants and 126 documents. Participants were undergraduate students enrolled in a programming-intensive software design course, who used the documents to learn professional software design know-how. We designed the field study to maximize its ecological validity and avoid interference with participants, as they should prioritize the course's learning objective over their participation in the study. We analyzed over 18 000 participant actions on the documents to assess the strengths and limitations of five presentation aspects of Casdoc that can be replicated in other presentation formats. Based on our results, we highlighted lessons learned about the design of code-oriented documentation formats. We also leveraged these findings and feedback received on preliminary versions of this work [53, 54] to improve Casdoc. We released a public set of learning resources for software design that use the improved version of Casdoc. This article makes the following contributions:

(1) the description of an interactive and non-linear format for software documents, which addresses common documentation issues (Section 2 and 5.4);
(2) an analysis of five relevant document design factors, based on prior work (Section 3);

(3) a complete methodology for the design of a field study that maximizes the ecological validity and reliability of the results in a context where the investigators have authority over participants (Section 4);
(4) the results of our study, synthesized into lessons learned for designing new software documentation formats (Section 5).

Replication. The material necessary to replicate our study is publicly available from the Casdoc project's web page, at https://www.cs.mcgill.ca/~martin/casdoc/. Readers can find on this page a free online service to convert annotated code files into the preliminary version of Casdoc used during the field study, with a detailed description of the annotation syntax. The page also includes links to the course textbook, which has a public companion website, and to the annotated code examples used in the study. The annotated code examples have been updated to the newest version of Casdoc, but their content is similar to what was available to participants during the field study.

2 THE CASDOC DOCUMENTATION FORMAT

Casdoc is a *presentation format* for online programming learning resources. Casdoc documents present a central code example, with additional explanations as interactive annotations. Authors create documents by writing regular source code files and inserting explanations in-place as code comments. The Casdoc *transformation tool* then converts the annotated code files into interactive web documents. Our implementation currently supports code examples written in the Java programming language.

Casdoc is designed for learning resources that focus on the implementation of programming concepts, such as programming forum posts and tutorials. It can demonstrate how to use a programming technology or the realization of programming concepts such as design patterns. In contrast, Casdoc is not intended for internal developer documentation and documents that focus on theoretical concepts.

2.1 Presentation Format

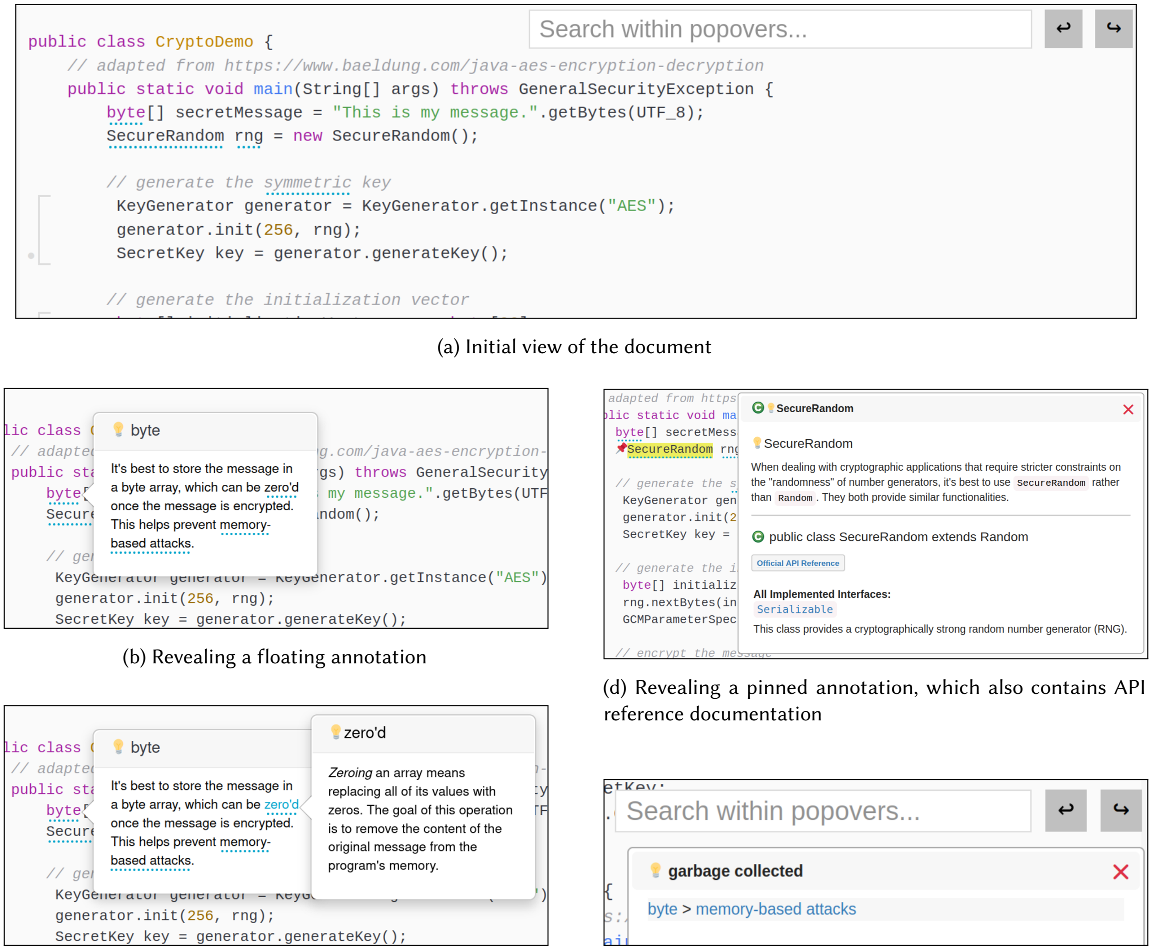
Figure 1 presents five views of a Casdoc document.[1] The initial view of the document shows only a central code example, which acts as the *root* of the document. For example, Figure 1a shows a code example that illustrates how to use Java's cryptography application programming interface (API) to encrypt a message.

Additional explanations of the root code example are placed in *annotations*. Annotations are interactive elements overlaid on top of the code example. They are hidden in the initial view of the document. Readers can selectively reveal the annotations relevant to them, then hide them again once they no longer need the information.

Each annotation contains some information about a specific code element, called its *anchor*. Anchors have visual *markers*, which indicate the presence of additional explanations to the reader. The anchor of an annotation can be any string of text on a single line (*inline anchor*) or any continuous set of lines (*block anchor*) in the code example. Figure 1b shows an annotation, anchored on the keyword `byte`, that explains why the original message is stored in a `byte` array as opposed to a `String` object. Some annotations can be associated with multiple anchors, for example when an important code elements appears multiple times.

The anchor of an annotation can also be a string of text *inside* another annotation, such as the mention of an important concept. In this case, the annotation that contains the anchor is the *parent* annotation, and the annotation that the anchor links to is a *nested* annotation. Figure 1c shows a nested annotation that defines the expression "zeroing

[1]The details of the format shown in Figure 1 is consistent with a preliminary version used in field testing. We introduced it first as a tool demonstration [53]. We made several improvements to the format, including visual modifications, based on the results of the study. Those improvements are described in Section 5.4. The description of Casdoc in this section applies to both versions of the format.



(a) Initial view of the document

(b) Revealing a floating annotation

(c) Revealing a second, nested floating annotation

(d) Revealing a pinned annotation, which also contains API reference documentation

(e) Secondary navigation tools available to readers

Fig. 1. Example of a Casdoc Document with Different Annotations Revealed

an array", which appears in the annotation about the byte array. Nested annotations can themselves contain other nested annotations.

Readers can view annotations in two forms. Hovering over an anchor reveals a *floating* annotation, which disappears when the reader leaves the area of the anchor and its annotation (Figure 1b). Clicking on the anchor *pins* the annotation, keeping it visible until the reader clicks again on the anchor (Figure 1d). Readers can move and resize pinned annotations.

Typical annotations come from the comments inserted by the document's author in the annotated code file. However, the Casdoc transformation tool automatically creates additional annotations with the official API reference documentation for standard Java types and their members (*Javadoc* annotations), anchored on the type or member's name in the code.[2] By contrast, annotations created by the document's author are referred to as *authored* annotations. If the anchor

[2]The anchor of Javadoc annotations are not indicated by markers, to avoid too many anchors and because the presence of the annotation is more predictable than arbitrary authored annotations.

```
// adapted from https://www.baeldung.com/java-aes-encryption-decryption
public static void main(String[] args) throws GeneralSecurityException {
    /*?
     * Type: Keyword
     * Anchor: byte
     * ---
     * It's best to store the message in a byte array, which can be zero'd
     * once the message is encrypted. This helps prevent memory-based attacks.
     *
     * +++
     *
     * Type: Internal
     * Anchor: zero'd
     * Parent: byte
     * ---
     * *Zeroing* an array means replacing all of its values with zeros.
     * The goal of this operation is to remove the content of the original
     * message from the program's memory.
     */
    byte[] secretMessage = "This is my message.".getBytes(UTF_8);
    /*?
     * Type: Keyword
     * Anchor: SecureRandom
     * ---
     * When dealing with cryptographic applications that require
```

Fig. 2. Source of the Two Annotations Shown in Figure 1c. The first annotation is anchored on the keyword byte in the code example. The anchor of the second annotation is the term *"zero'd"* in the first annotation. The first annotation also included another nested annotation, which was removed from this figure to preserve space.

of a Javadoc annotation overlaps with the anchor of an authored annotation, the two annotations are combined into a single one, with the two fragments clearly separated. Figure 1d shows an annotation that contains both the rationale for using the SecureRandom class to generate numbers, and the reference documentation for that class.

To help readers orient themselves across the graph of annotations, Casdoc includes several visual aids and navigation tools. When an annotation is pinned, a pin icon appears beside its anchor, and the anchor is highlighted when the reader hovers over the annotation (Figure 1d). Pinned nested annotations show a *breadcrumb trail* to indicate their parents and allow readers to open them (Figure 1e). Readers can also use a custom *search bar* to search among the content of all annotations.[3] Finally, readers can *undo* and *redo* pinning and unpinning actions, e.g., in case they accidentally close a nested annotation and forgot where the anchor was.

2.2 Authoring Process

To create a Casdoc document, an author starts by providing the root code example in a new Java file. The author then inserts annotations in code comments next to their anchors. Embedding annotations in a code file provides a format familiar to programmers. It ensures that the root document can be created and maintained using common development tools, such as integrated development environments (IDEs) and version control systems (e.g., Git). Each Java file will be converted into a separate Casdoc document.

Authors use a special syntax to distinguish code comments that contain annotations from regular comments to keep in the final document. Figure 2 shows the comment that declared the two annotations shown in Figure 1c as an example.

Each annotation comment is enclosed between the sequences /*? and */, and may contain multiple annotations separated by lines containing three or more plus signs (+). Each annotation starts with the declaration of its anchor as a series of lines containing colon-separated key-value pairs. The Type key indicates the type of anchor to declare: Keyword

[3]The native search feature of web browsers cannot reveal or find text in hidden annotations.

for top-level inline anchors, Internal for nested inline anchors, and Block for block anchors. If the anchor is nested, the Parent key identifies the parent annotation by its anchor (inline annotations) or title (block annotations). The Anchor key declares the substring to use as an inline anchor. The substring must appear in the line immediately following the block comment (top-level anchors) or in the content of the parent annotation (nested anchors). Block anchors use the Range key to declare how many lines the block anchor spans, starting at the line immediately following the block comment. Authors can also use additional key-value pairs to declare some metadata of the annotation, such as its title using the Title key (optional for inline annotations, but required for block annotations) and hyperlinks to reference material using the URL key.

The series of key-value pairs ends with a line containing three or more dashes (-). Below this line, the author declares the content of the annotation. Authors can use the Markdown syntax to create visually rich annotations [27].[4]

The special syntax for declaring annotations does not interfere with the original Java code. Therefore, the annotated files can be validated for syntax and symbol resolution by any suitable Java compiler. After inserting the desired annotations, the author uses the transformation tool to convert the annotated Java file into the final Casdoc web document.

2.3 Implementation

Casdoc documents are self-contained. The HTML file generated by the transformation tool contains all declared annotations, using dedicated HTML elements to identify them and their anchors. The visual elements and interactive aspects of the format are implemented with client-side CSS and JavaScript assets, which themselves only rely on mature libraries.[5] Hence, Casdoc documents can be viewed in any software that supports standard web technologies and can be deployed easily without requiring a complex server infrastructure.

The transformation tool is implemented as a Java program. It relies on a Java-specific parsing and symbol resolution library to extract custom annotations from code comments and Javadoc annotations related to standard types and members. A preliminary version of the tool is available as a free online service from the project web page.[6]

3 KEY PROPERTIES OF CASDOC

Documentation formats exhibit characteristics that vary across numerous dimensions, such as the length of code examples, the interplay between text, figures, and code, or the use of external resources as integral or peripheral information sources [5, 28]. The creation of Casdoc involved many decisions, from core design principles to technical implementation details. Not all of those decisions, however, have the same impact on the ways readers find information in documents. We identified five properties of Casdoc that are key components of the format. Casdoc documents

(1) show code examples before other types of content,
(2) gradually reveal information,
(3) split information into small fragments,
(4) use explicit hints to help readers navigate within the document, and
(5) integrate content from external sources.

[4]This description of the annotation syntax is consistent with the revised version of Casdoc, described in Section 5.4. The original syntax used to create the documents described in this article followed similar principles, but it was harder to parse and limited the potential for future expansions. The original syntax is precisely defined on the Casdoc project website (see footnote 6). The differences between the two versions did not affect the format of the generated HTML documents.
[5]The web assets can optionally be embedded in the Casdoc document, to make it a truly self-contained HTML file.
[6]https://www.cs.mcgill.ca/~martin/casdoc/

Table 1. Presence of the Five Properties in Documents from Various Sources

Document Source **Link**	**Code-First**	**Gradual Reveal**	**Small Fragments**	**Explicit Hints**	**External Content**
Casdoc https://www.cs.mcgill.ca/~martin/casdoc/	✓	✓	✓	✓	✓
Oracle's Java Tutorials https://docs.oracle.com/javase/tutorial/java/index.html	-	-	-	-	-
Java API documentation (Javadoc) https://docs.oracle.com/en/java/javase/17/docs/api/	-	-	✓	✓	-
Stack Overflow https://stackoverflow.com/questions	-	-	✓	-	-
Android Developer Guides https://developer.android.com/topic/architecture	-	✓	-	✓	-
Amazon API Gateway's FAQs https://aws.amazon.com/api-gateway/faqs/	-	✓	✓	-	-
R Cookbook https://rc2e.com/	✓	-	✓	-	-
Codelets [55] https://dl.acm.org/doi/10.1145/2207676.2208664	✓	✓	✓	✓	-
Adamite [33] https://adamite.netlify.app/	-	✓	✓	✓	-
SISE [72] https://dl.acm.org/doi/10.1145/2884781.2884800	-	✓	✓	-	✓

The properties are informed by prior work on programmer information needs and reading behavior. Each property thus corresponds to a hypothesis, namely that the property will help readers locate the information they need within a document. We used these properties to scope our evaluation of Casdoc. This perspective focuses our findings on aspects of documentation that can be found in other existing formats or that can be used to design new ones.

We present each property with the prior work that supports it, its realization in Casdoc, and other examples of the property found in existing documentation. We also discuss potential limitations of the property, or contexts in which it may be detrimental to a document's quality. Table 1 shows an overview of the properties across a sample of documentation sources and formats. Those examples demonstrate alternative implementations of the properties. We note that the presence or absence of a property does not correlate with the overall quality or usefulness of the documents. In particular, the official Java tutorial documents (second row), despite being of high quality, do not exhibit any of the studied properties. We selected all resources as examples of good documentation.

We discuss these properties as they apply within the context of a single document, i.e., a single web page. The organization of documents within a set is outside the scope of this work.

3.1 Code-First Presentation

> The document format guides readers toward high-quality code examples that they can use to understand a concrete application of software development technologies.

Tutorial authors recognize the importance of good code examples [28] and many of the documentation retrieval and synthesis techniques focus on code examples as the main source of information (e.g., [20, 80]). Code examples capture concrete solutions that anchor the discussion of more abstract concepts or guidelines related to the code. A code example is also a useful starting point for programmers to copy and adapt to accomplish their task. For these reasons, programmers commonly choose to first read the code examples of a tutorial, and only refer to surrounding text if they need more information [12]. As a result, documents without code examples, or with code examples that are too simple, are viewed as less helpful by programmers [51, 61]. Even documentation platforms that encourage the use of code examples, such as Stack Overflow, may not use a format that draws the attention of readers to them, and researchers have proposed code extraction and synthesis approaches to address this limitation [25]. The benefits of code examples in learning resources are similar, to an extent, to the benefits of video tutorials: they allow the audience to follow along a concrete application of the abstract concepts being discussed [46].

Implementation. Casdoc supports a code-first presentation by anchoring the hierarchy of annotations in a complete and compilable code example.

Alternative Implementations. Some tutorials are accompanied by curated sets of standalone examples, intended to illustrate the notions described in the tutorial within solutions for more complex scenarios.[7] Other online resources, such as GitHub Gist,[8] are themselves databases of code examples, often with a minimal description of the example's purpose, which can be used on their own or in combination with other documents. Programming "cookbooks" are a more structured version of code example databases.[9] They typically focus on implementation solutions that the reader can adapt to perform common tasks with a technology. Some online learning platforms also guide readers through the implementation of a small program as the main learning activity.[10] However, although such platforms place a higher importance on code, some readers may prefer to see the complete program first, instead of going through each step of the guide. Oney and Brandt proposed to embed documentation in shareable code fragments, called Codelets [55]. Programmers create Codelets as an HTML document using specific tags to identify links between the code example and its related documentation. Although their idea aims at helping programmers integrate code examples found on the web, rather than as a learning resource directly, Oney and Brandt mention the pedagogical potential of Codelets.

Trade-offs. Code examples alone are often insufficient to describe complex programming tasks. Documents that present code first should not entirely omit accompanying explanations, which can help the reader adapt the code example to their situation, distinguish important parts of the code from peripheral elements, or learn related concepts. For example, Stack Overflow answers that contain only code often receive downvotes or edit requests to add some explanation of the code [51].

3.2 Gradual Reveal

> The document format reveals only a small part of the content at a time, letting the reader understand one fragment before showing the next.

Being overly verbose and containing insufficient information are two common, yet conflicting, issues of documentation [3]. Including more information in a document is necessary when the audience is large and varied, as it is often the case for software documents. However, too much content can have a detrimental effect if it increases the time and effort each reader takes to find the information they need. When readers spend, or estimate that they would spend, too

[7] E.g., https://www.tutorialspoint.com/javaexamples/index.htm and https://www.w3schools.com/java/java_examples.asp
[8] https://gist.github.com/
[9] E.g., the Python [9] or R [42] cookbooks
[10] E.g., Google Codelabs, https://codelabs.developers.google.com/

much effort to find the parts of a document they need, they are likely to look for another document [43, 81]. In their idea of the "Minimal Manual", Carroll et al. suggest to "slash the verbiage" [14]: technical writers should reduce the length of manuals by removing redundant and superfluous parts, to avoid readers misusing the documents or missing crucial information. Crowd-sourced documentation platforms in particular, such as Stack Overflow, can accumulate an overwhelming amount of content on popular topics. They must find effective ways to emphasize the most important information to readers [82]. Exposing readers to only parts of a document at a time is an alternative solution to this conflict between completeness and verbosity.

Implementation. Casdoc gradually reveals its content through annotations in floating popovers or pinned dialogs. The initial view of a Casdoc document shows only the code example, and the reader chooses which annotations to reveal by interacting with their anchors.

Alternative Implementations. Collapsible HTML components can be used to allow readers to choose which information to reveal in a document.[11] Tabbed containers can be useful to include multiple variants of the same content fragment in a document without increasing its visual weight. They can show, for example, information to accomplish a task with alternative technologies, allowing readers to select the technology that is relevant to them.[12]

Trade-offs. Revealing information gradually inherently relies on a format where readers modify the document. With Casdoc, this feature additionally involve a non-linear organization of information. Requiring readers to make decisions about which content to read can increase their cognitive load [21]. However, this effect is mitigated if the content of a document does not itself have an inherent linear order [84]. As another consequence, dynamic documents may be harder to consistently implement or adapt across software applications and viewing devices. For example, the Casdoc format is designed for mouse interaction in desktop web browsers, and does not support well mobile devices, touch-based interactions, or printing. This single supported context limits the usability of Casdoc documents. Furthermore, dynamic documents are not well suited for long-term archival purposes, unless the viewing technology is archived with them. Thus, it may be useful to produce a static version of dynamic documents as a replacement in situations that do not support user interactions.

3.3 Small Fragments

> The document format presents its content as a series of concise fragments that each convey a single self-contained idea.

It is common for programmers to read a document out of sequence [12]: they may look for a specific section related to their needs, skip information that they already know, or go back to an earlier point in the document to find background information about a concept. A set of concise and decoupled fragments supports such reading behavior. In contrast, documents composed of vaguely bounded and highly dependent fragments force their readers to read larger sections to contextualize and understand the information they seek, which can create a feeling of verbosity. Additionally, identifying clear fragments in a document can facilitate the reuse of the content into other documentation systems (e.g., [20, 35, 80]) or in integrated development environments (e.g., [55, 57]). This reuse scenario expands the value of the document's information beyond its original purpose.

[11] E.g., the FAQs document of Amazon API Gateway uses a collapsible element for the answer of each question: https://aws.amazon.com/api-gateway/faqs/
[12] E.g., Android Developer Guides uses this strategy to show equivalent code examples either in Kotlin or Java: https://developer.android.com/guide

Implementation. Casdoc's annotations encourage authors to partition the information into concise fragments that will be presented in small popovers and dialogs. Annotations can link to further supporting explanations in nested annotations, but they should present a complete idea by themselves.

Alternative Implementations. Non-interactive formats can also be organized as small fragments. For example, API reference documentation typically contains one fragment per API type or member.[13] Readers are not expected to read the entire API documentation to understand the fragment about a particular element. Question and Answer (Q&A) forums often exhibit this property, as answers are typically created as independent fragments.[14]

Trade-offs. Documents that appear too fragmented can irritate readers [74]. Fragmentation can lead to frustration when readers do not know how to find a fragment of interest, or when they need to gather multiple fragments scattered across a document to answer a query. To prevent this problem, authors must carefully assess the inherent relationships between fragments and replicate these relationships in the document's structure. Dividing the content of a document into small fragments can also break its narrative flow or disorient readers that do not know what information they must look for. Thus, this property may be detrimental in some contexts, such as online courses for a homogeneous novice audience.

3.4 Explicit Hints

> The documentation format includes explicit hints, distinct from textual cues, to help readers understand and navigate the structure of a document.

Navigating within the content of a document is an important aspect of information search [56]. Given the amount of web resources readily available and indexed by search engines, readers have a strong incentive to look for other documents if they do not find the information they seek quickly in the current one. Visual hints of the organization and content of a document reduce the cost of within-document navigation and provide a sense of location and control [71]. Navigation features (e.g., link previews, similar to Casdoc's floating annotations) can also help reduce the potential cognitive load incurred by a fragmented, non-linear format [4, 21]. This property is more incremental than the previous ones. Multiple types of visual hints can be incrementally added to a format to reveal complementary aspects of the organization of information.

Implementation. Casdoc uses markers to indicate the presence of annotations related to a code element or to a concept. Annotations that contain only Javadoc information, however, do not have markers as Casdoc systematically adds such annotations to all API elements in the standard Java libraries, making their presence more predictable than authored annotations. Casdoc also uses indicators such as "pin" icons, breadcrumbs, and highlighting to identify the anchor of a pinned dialog.

Alternative Implementations. The Adamite annotation tool uses visual markers to highlight the parts of a document that have been annotated by readers [33]. The XCoS code search approach presents information related to different aspects of the query (e.g., non functional requirements) to help users navigate the list of results [78]. Although those hints are text-based, they constitute a navigation structure distinct from the main list of results. Existing documents also include recurrent types of alternative hints. For example, a table of contents that remains visible and indicates the current position of a reader as they scroll within a page is useful to convey a sense of location within an overview of

[13] E.g., the Java API reference documentation (Javadoc), https://docs.oracle.com/en/java/javase/17/docs/api/index.html
[14] For example, the popular Stack Overflow programming forum, https://stackoverflow.com/questions/

the document.[15] API reference documentation uses hyperlinks to relate relevant fragments, such as a function to its parameter and return types.[16] Although hyperlinks are a common feature of many websites, the predictable nature of a link's target in reference documentation makes them an effective mechanism to navigate its structure, as opposed to the arbitrary links in typical documents.

Trade-offs. Structural cues integrated in the main text of a document must be used sparingly. They can bloat the content and dilute its relevant information. Ideally, explicit hints should be clearly separated from the document's content, so that the reader can ignore them once they reached the information they sought. Alternatively, hints that rely only on non-textual elements, such as Casdoc's markers, can easily be distinguished from the content. However, the hints' purpose must be intuitive, or they risk confusing the readers. For example, complex visualizations of a document's overview can increase the cognitive load of readers, negating its intended benefits [21]. Non-textual hints can also limit the accessibility of a document for some readers. For example, screen readers will not detect Casdoc's markers. Thus, visual hints should be complemented with other navigation cues, possibly embedded in the HTML tag attributes.

3.5 External Content

> The document format provides a systematic way to integrate information from external sources within its original content without corrupting or misappropriating either source of information.

The extensive prior work on documentation generation and information retrieval (e.g., [41, 57, 72, 80]) constitutes a valuable opportunity to increase the information coverage of a document. Formats should be designed to leverage these approaches to reduce the effort of multiple authors documenting similar technologies, similarly to how software development evolved to promote the reuse of software packages (especially when well documented).

Implementation. Casdoc automatically integrates API reference documentation as additional annotations. These third-party annotations are identified by special icons and contain a link to the information's source. When the anchor of a third-party annotation overlaps with the anchor of an authored annotation, the two are concatenated into a single annotation that clearly distinguishes its two parts.

Alternative Implementations. The SISE tool designed by Treude and Robillard integrates information fragments from the Stack Overflow forum at the top of API reference documentation pages [72]. The imported information is presented in a rectangle overlay that is kept distinct from the API reference, and contains links to the original Stack Overflow posts. The browser extension developed to showcase the Baker traceability recovery tool uses an approach similar to Casdoc [68]. It inserts annotations in code examples on Stack Overflow that readers reveal them by hovering over API elements. Contrary to Casdoc, the annotation contains links to other documents, such as the API reference documentation or related Stack Overflow questions, instead of importing the external content.

Trade-offs. The trustworthiness, authoritativeness, and tone of imported content can differ from the document's original content and vary between external sources. Thus, including content from various sources can create jarring changes for the reader, which can affect the perceived qualities of the original content. Clearly identifying the provenance of external content can mitigate these issues. Attributing proper credit is also an ethical and sometimes legal requirement.

[15] The Android Developer Guides use such interactive tables of contents, https://developer.android.com/guide/components/fundamentals
[16] For example, Java's API reference, https://docs.oracle.com/en/java/javase/17/docs/api/

4 STUDY DESIGN

We evaluated Casdoc in a field study with undergraduate students enrolled in a software design course. Throughout the course, participants had access to a suite of Casdoc documents that complemented the course material. We analyzed how they navigated within the content of each document to assess the strengths and limitations of Casdoc.

The goal of our study was to observe how programmers use the different features of Casdoc. We gathered empirical evidence about the usage of Casdoc's features to test the value of Casdoc as an addition to the documentation format design space and to generate hypotheses to better understand the needs and behavior of readers, within the context of dynamic documents. We also aimed to collect actionable metrics to improve Casdoc, or identify which popular features deserve more attention. We used the following research questions to guide our study.

RQ1 Is Casdoc a suitable format for creating learning resources for programmers?

We addressed this question by comparing the adoption of Casdoc to that of an alternative baseline format. The objective of this question is not to argue for the superiority of either format, but rather to contextualize the usage Casdoc relative to a known practical format.

RQ2.1-RQ2.5 What is the impact of [property 1-5] on the navigation behavior of the readers of a document?

For these questions, we instrumented the generated Casdoc documents to log events when participants interacted with the different features. We correlated the information seen by participants to the five properties described in Section 3, to assess whether the properties can help navigate the content of a document.

4.1 Research Method

Our study falls within the *field experiment* category of Stol and Fitzgerald's framework of research methods [67]. This category includes studies conducted in pre-existing environments, but that modify at least one aspect of the environment. It does not require the comparison of an experimental condition to a control condition. We designed the study to prioritize the ecological validity of the results. We conducted our investigation within a natural setting, i.e., a university course, but manipulated the environment to introduce Casdoc documents. This strategy favors the realism of the setting, while allowing the introduction of new elements, such as the Casdoc format, that do not exist in a purely natural setting.

Although participants could freely choose and change the documentation format they used, the analysis focuses on the data related to the experimental condition, as opposed to a comparison of both conditions. Beyond the field study, our investigation also integrates some aspects of action research, as we continued to improve the Casdoc format based on our findings, to integrate the new documents as a permanent part of the course material for future students (see Section 5.4).

4.2 Participants

The field study took place during two consecutive sections of a third-year undergraduate course on software design with an important programming component. All students enrolled in the course could choose to participate in the study by agreeing to a consent form for the collection of their interaction data.[17]

[17] This study was approved by the Research Ethics Board Office of McGill University, file number 21-06-007.

Students are a subgroup of the target audience for Casdoc, i.e., programmers who are learning software development concepts and the usage of some libraries. Although we do not claim that this sample is representative of all programmers, the participants did not act as proxy for a different population.

Both authors had a teaching role in the first section. This familiarity with the course was crucial to create relevant Casdoc documents. The authors were not involved with the second section.

Given that the investigators had authority over participants of the first section, participants remained completely anonymous throughout both sections of the study. This anonymity was important to avoid an unintentional pressure on students to participate in the study or use a format if they did not feel comfortable. For a similar reason, we did not offer any compensation to participants, other than the potential benefits of the new format. Other forms of compensation were also impractical because the anonymity of participants was maintained when obtaining informed consent. The consent form was delivered on the website that served the documents. A client-side script automatically enabled data collection when participants indicated their consent in the form. The form did not require or collect any personal information.

Consequently, we did not collect demographic information to measure specific properties of the sample. However, the population from which the sample is taken is well known. Senior undergraduate students in computer science consists predominantly of young adults with only a few years of programming experience, with a minority having previously done industry internships. Before registering for the software design course, students are expected to be familiar with the programming language of the course, Java, and its standard library, but not necessarily with advanced concepts.

4.3 Documents

We used the content of the companion website of the course's textbook [60] to create the corpus of Casdoc documents.[18] The website contains three types of documents, namely lists of exercises, descriptions of their solutions in prose, and 126 code examples: 72 of them implement code described in the textbook (i.e., *chapter code*) and the other 54 implement solutions to the exercises (i.e., *solution code*).

We converted each code example to the Casdoc format and inserted additional explanations as annotations. The original code examples sometimes contained code comments. We retained those comments in the converted documents, rather than transforming them into further annotations.

We authored annotations based on our experience of past students' needs rather than a systematic content generation approach. The additional explanations described, for example, aspects of the Java language syntax (e.g., the role of the `assert` keyword), details about library methods (e.g., the difference between JUnit's `assertEquals` and `assertSame`), design details specific to the code examples (e.g., the rationale for using private fields with accessor methods), or definitions of software design concepts (e.g., design by contract). Some annotations contained other types of information (e.g., interesting anecdotes or alternative implementations) or information related to multiple of these categories. In total, we added annotations to 70 of the 126 documents. An additional 31 documents contained only Javadoc annotations, and 25 documents contained no annotation. Examples of documents with no authored annotations include code examples copied from a previous chapter and simple scaffolding code that already contained sufficient code comments. Annotated documents contained up to 27 annotations (median: 4, average: 6.0).

[18] https://github.com/prmr/DesignBook

Table 2. Types of Anchors Used for the Authored Annotations

Anchor	Frequency
Inline anchors	**318**
Declared class/method/parameter/variable	70
Library API	42
Java keyword	36
Other code	30
Concept mention	51
Other text	89
Block anchors	**99**
Single line	25
Multiple lines	74

The resulting Casdoc documents contained 417 authored annotations. Of those annotations, 166 had a top-level inline anchor, 99 had a top-level block anchor, and 152 had a nested inline anchor. Table 2 presents a fine-grained overview of the types of anchors. Most inline anchors in the code were on the identifier of a class, method, parameter or variable declaration. We also often used types and methods from JUnit, JavaFX, and the Java standard libraries, as well as design and computing concepts as anchors. The documents contained a small fraction of duplicate annotations (51 out of 417, or 12.2%), typically when the same design concept appeared in multiple documents within the same chapter. We did not carry over annotations across chapters.

In addition to the Casdoc documents, we converted the annotated code examples to a static baseline format. This format included all authored annotations as code comments, but it did not include the API reference documentation to avoid unreasonably large comments. This information is, however, easily accessible via the students' integrated development environments (IDEs) and via the official Java documentation website. We did not modify the exercise or solutions, which consist mostly of text.

The code examples were available on a public website dedicated to the study. The website showed the first document in the Casdoc format to all participants to encourage them to try the new format and provide a consistent user experience. When viewing a document, participants could change between the two formats whenever they wanted to. The website stored the last format used in a browser cookie to open the next document in the same format.

The study website initially contained only the annotated code examples. After observing a low study participation rate during the first section, we added the exercises and solutions, unmodified, to the website. Following this change, the retention rate of participants increased for the second section.

4.4 Data Collection Infrastructure

We instrumented the documents to record traces of the participants' activity. Asynchronous client-side JavaScript functions created the interaction events and sent them to an HTTP POST endpoint of a dedicated data collection server. This logging mechanism was transparent to the participants. The baseline documents did not generate interaction events as they lacked interactive features, but the server recorded document requests in either format and requests to change the format.

The study website did not require identification or authentication to preserve the anonymity of participants. Instead, it stored two HTTP cookies in each participant's browser, in addition to the format-related cookie. Upon consent,

Table 3. Events Collected During the Field Study

Event	Origin	IDs	Details
Visit any page[a]	server	D	
Consent to study	server	P/S	
Withdraw consent	server	P	
Start new session	server	P/S	
Open code example	server	P/S/D	format
Change format	server	P/S/D	new format
Open/Close annotation	client	P/S/D	annotation ID
Interact with marker	client	P/S/D	marker ID
Use search widget	client	P/S/D	query; selection(s)
Use navigation widgets	client	P/S/D	effect on annotations

[a]This event was only collected during the second section of the course.

participants received a randomly generated 64-bit integer in a persistent cookie (i.e., the *participant ID*). The website also sent a second random integer in a session cookie (i.e., the *session ID*), which was reset every time the browser was reopened.

Table 3 summarizes the types of events we collected. The first six types of events are generated by the website, whereas the last four types are generated by JavaScript functions and sent through the HTTP POST endpoint. For each event, the website stored the type of event and a timestamp, as well as the IDs of the participant (P), session (S), or document (D) and the additional details described in the last column.

The first type of event relates to information about the study website, rather than about participants' activity. The events capture each document request, including requests for documents other than the code examples. Contrary to the other events, the first type of event did not include any information about the origin of the request. They also did not allow us to correlate multiple events from the same user to identify patterns. We modified the server to record these events for the second section, both to monitor the website's status and to potentially detect tampering attempts.[19] We also compared the number of requests handled by the website to the participant-related events to estimate the sampling bias (see Section 4.7).

To keep the data collection procedure minimally intrusive to participants, we did not rely on tools such as pop-up dialogs or surveys to gather the feedback of participants. Thus, we did not collect insights about the subjective perceptions of participants on Casdoc, a concession to reduce the threat of observer effect.

4.5 Data Preparation

Table 4 gives an overview of the data we collected. In total, 326 participants generated over 18 000 interaction events. They consulted the 126 code examples a total of 7338 times.

We reassembled the flat list of events into a meaningful structure to analyze our results. The actions of each participant are split into *sessions*, i.e., a period of continuous usage of the website. During a session, a participant *viewed code example documents*. Participants performed different actions on code examples, such as *viewing an annotation* and *using the search widget*. We distinguish between annotations that we *authored* and those with imported *Javadoc* content.

[19]We did not observe any unusual access patterns, except for a large number of requests to the website's home page: there were almost six times as many requests to the home page as the number of requests to all code examples combined. These requests could be due to web crawling or server maintenance bots.

Table 4. Summary Statistics of the Collected Data

Property	Section 1	Section 2	Total
Study length (days)	104	102	206
All document requests[a]	–	19 594	–
Code example requests[a]	–	14 644	–
Enrolled students	165	321	486
Participants	124	202	326
Sessions	176	541	717
Opened code examples	827	6511	7338
Logged interactions	2795	15 570	18 365

[a]including from non participating students

Based on a preliminary inspection of the data, we considered all events performed within 15 minutes of consenting to the study as part of a *learning period*. We excluded the data of all participants who did not interact with the website beyond their learning period.

We found that the session cookie was unreliable to track continuous usage. Some participants rarely closed their web browser, creating sessions that spanned many days or weeks. We split long sessions between events separated by at least two hours.[20] Within each session, opening a document initiates a new code example view. Documents that remained opened through artificially split sessions constitute new code example views in the second part of the session only if the participant performed any action on the document.

We grouped successive events associated with the same Casdoc annotation as a single annotation view action. Each annotation view starts with zero or more hovering events, optionally followed by a pin event, and a final optional unpin event. We grouped together multiple hovering events if they were less than five seconds apart, and ignored hovering actions that lasted less than one second.

As each keystroke in the search widget generated a new event, we grouped all events that incrementally built towards a single search query, as well as subsequent interactions with the search results, as a single search action. Each use of the breadcrumbs and the undo and redo buttons constitutes a separate action.

4.6 Study Design Trade-Offs

There is an inherent trade-off between the realism of, and control over, the study setting. As the field study favors realism, we could not control when or for how long participants used the documents. Field studies also lack the control of confounding factors that the sterile environment of laboratory experiments provides. Thus, the decision to conduct a field experiment limited the precision of our measurements and generalizability of our results [67], but produced concrete insights that are directly applicable to an existing context. These insights led to the improvement of Casdoc for future students enrolled in the course.

Another early decision point was the choice of the environment in which to conduct the study. We chose to study students enrolled in a university course. Alternative options included looking for programmers outside our organization, such as professional developers, or using remote experimentation platforms such as Prolific.[21] The effort involved in

[20]We chose the threshold of two hours based on the distribution of time between two consecutive events with the same session cookie. Nevertheless, as we did not observe a significant drop in the distribution, this threshold is only approximative. When possible, we avoid relying on sessions in our analysis.
[21]https://www.prolific.co/

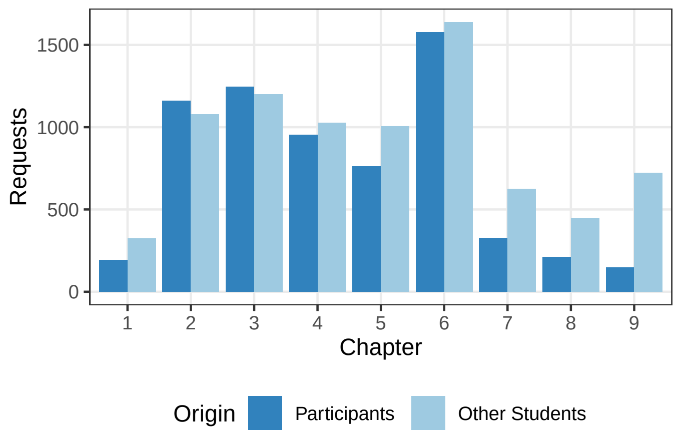
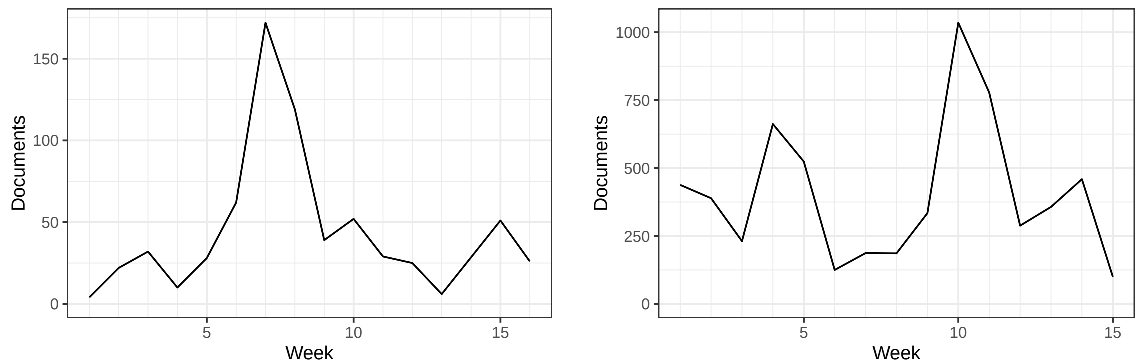


Fig. 3. Code Example Requests by Chapter from Participants and Other Website Users During Section 2

recruiting participants for a long-term study (several months) and creating a realistic environment in which participants would need learning resources was a deciding factor for choosing the university course. A consequence of this decision was the need to mitigate potential pressures on students. We thus designed the data collection to be anonymous and minimally intrusive, at the cost of collecting only quantitative usage data. Additionally, the threat of demand characteristics bias increased as students knew that Casdoc was designed by their instructor and a teaching assistant during the first section. A benefit of this context, however, was our ability to author relevant documents for students. Recruiting students as participants also narrowed down the sampling frame of our study. Thus, our results are specific to a well-defined subset of Casdoc's target audience.

The number and choice of document formats to compare was also an important decision. Alternative formats include presenting the additional explanations in a narrative text above, below, or interleaved with the code example, as well as presenting a varying number of explanations in static documents. Offering more formats to participants can help contextualize our observations. However, each format requires a considerable effort to produce, and too many formats can overwhelm participants. We chose to offer one baseline format to have at least one point of comparison for Casdoc. We selected static, commented code examples for the baseline as it is conceptually the closest to Casdoc. We did not vary the content of documents to avoid students missing some relevant information due to their choice of format.

Regarding the collection of events, there is a trade-off between the reliability of the events and the quality of the user experience. Relying on cookies and asynchronous client-side functions increase the risk of lost or corrupted data, e.g., if a browser automatically deletes cookies. The website's HTTP POST endpoint was also vulnerable to potential attacks from malicious actors, who may try to send fake events. We accepted these risks to improve the user experience and honor our responsibility to create a suitable learning environment for students in the course. To limit and detect the generation of corrupted events, we ensured that key events were generated by the website, such as new document requests. We also devised a strategy in which the type of event in the client-side scripts would be encrypted based on the session and participant IDs to detect fake events. We found no inconsistency in the collected data.



(a) Section 1 (b) Section 2

Fig. 4. Number of Code Examples (Documents) Accessed by Participants During Each Section of the Course

4.7 Sampling Bias

When recording all document requests received by the study website during the second section of the course, we observed that the majority of requests (55.1%) were not made by participants. As participation in the study was voluntary, there is the risk of a sampling bias in our results. For example, students who are less favorable to trying new technologies may decide to only use the more familiar baseline format and forget to consent to participate in the study.[22] To assess the magnitude of the sampling bias, we investigated whether there was a notable difference between the events generated by participants and the requests received by the study website that did not match participants' events. Figure 3 compares the requests for code examples for each chapter. We observe that the differences are relatively small. A Pearson's χ^2 test confirms that the differences are statistically significant ($p < 10^{-15}$), but the effect size is small (Cramer's $V = 0.18$). The same analysis, but comparing requests by week rather than by chapter, return similar results (Pearson's χ^2 test: $p < 10^{-15}$; Cramer's $V = 0.19$). Thus, although we cannot exclude the effect of a sampling bias on our results, there is no evidence of considerable differences between the sample and the target population.

5 RESULTS

Figure 4 shows a timeline of the participants' activity through the study, and Table 5 presents an overview of the main study artifacts and observations. Although we excluded 122 short-term participants (37.4%, see Section 4.5), we retained interaction data related to 6770 document views by the others. As participants viewed the large majority of documents (96.1%) in the Casdoc format, the data shows clear usage patterns for the different features of Casdoc, but only limited insights into situations that Casdoc does not support well. We present these patterns in Section 5.1, and lessons we learned about the design of Casdoc in Section 5.2. The analysis focuses on the data collected during the second section of the course, as the study conditions improved from the first section: (i) the investigators did not have a teaching role, (ii) all code examples were available from the beginning of the course, and (iii) we fixed limitations of the study website (see Section 4.4). We used the data from the first section to triangulate the patterns observed during the second section (see Section 5.3). The study findings informed several improvements to Casdoc, which we detail in Section 5.4.

[22]Students had to consent to the study to use the Casdoc format, as the instrumented client-side functions would generate interaction events.

Table 5. Summary Statistics of the Data After Preprocessing

Property	Section 1	Section 2	Total
Participants	54	150	204
Sessions	155	1060	1215
Unique code examples	123[a]	126	126
Code example views by all participants	677	6093	6770
Code example views in Casdoc format[b]	670	5836	6506
Unique authored annotations	417	417	417

[a]For technical reasons, three code examples were not available during the first course section. We fixed this issue for the second section.
[b]Excluding views during which the participant changed format

5.1 Casdoc Usage Patterns

Table 6 presents the detailed data collected during the field study, grouped according to our research questions.

To interpret the usage patterns that we observed, we rely on the assumption that the usage of a feature indicates the usefulness that participants perceive in this feature. This assumption is justified by the motivations of participants. As students, we assume that their primary motivation for using learning resources is to learn the concepts taught and pass the course. As the target of the study, Casdoc, is evident to participants, there is a risk of demand characteristics bias, i.e., participants changing their behavior based on what they believe the investigators expect. Although this bias is impossible to avoid, using low-quality resources would directly interfere with their primary learning objectives. Furthermore, the absence of compensation and the extended duration of the study limited the risk of participants using the documents for short-term rationales unrelated to their usefulness. The anonymity of participants and the absence of interaction with the investigators when using the study's instrument also mitigated the threat of participants trying to please the investigators. Finally, excluding the initial learning period for each participant limited the impact of interaction patterns caused by the initial exploration of an unfamiliar format. Nevertheless, in our discussion of the results, we identify interpretations that rely on this assumption in italics and with the [HYP] label to indicate their status as hypotheses.

RQ1. Viability of Casdoc. The majority of participants (129 of 150, or 86.0%) used only the Casdoc format throughout the course. Among the 21 participants who tried both formats, only five (24%) retained the baseline until the end of the course. Most of the participants who reverted to Casdoc did so within the same session. We investigated the type of the documents (i.e., chapter code or solution code) for which participants changed the format, number of annotations they contained, and the number of documents the participants had used before. However, there was no clear trend. For example, some participants switched to the baseline on their first document after the learning period, whereas others only tried the baseline after having already read over 80 documents.

[HYP] The frequent use of Casdoc compared to the baseline is evidence that Casdoc is already a valuable addition to the documentation design space. More generally, it also shows the openness of participants to try new approaches to format documents, despite the need to develop new navigation patterns.

RQ2.1. Code-First Presentation. We used the study website traffic information to assess the value of code-oriented documents, as both Casdoc and the baseline formats focus on a complete code example.[23] We compared the number

[23]This data originates from all users of the website as a group, within which individuals cannot be distinguished.

Table 6. Metrics and Results by Research Question

RQ	Question/Metric	Section 1	Section 2	Total
1	*Viability of Casdoc*			
	All participants	54	150	204
	Participants who used only Casdoc	49	129	178
	Participants who tried the baseline format	5	21	26
	… only during the learning phase	1	5	6
	… for only one document after the learning phase	2	8	10
	… for only one session (2+ documents) after the learning phase	1	3	4
	… for multiple sessions	1	5	6
	Participants who changed to the baseline format more than once	0	3	3
	Participants who kept the baseline format until the end	1	5	6
Finding: *Most participants only used Casdoc. Most of those who tried both formats changed back to Casdoc within the same session.*				
2.1	*Code-First Presentation*			
	Server-side code example requests	—	14 644	—
	… chapter code	—	8857	—
	… solution code	—	5787	—
	Server-side exercise statement requests	—	2539	—
	Server-side solution description requests	—	2411	—
	Solution code to description requests ratio	—	2.4	—
	Average number of links to solution code per solution description	—	3.8	—
Finding: *Participants consistently used documents centered around code examples to complement the concise information found in other documents.*				
2.2	*Reveal Information Gradually*			
	Annotation views	356	1889	2245
	Participants who viewed at least one annotation	35	115	150
	% annotated document views with 1+ annotation view(s)	18.8%	15.6%	15.9%
	% markers in the code example interacted with (average, by participant)	9.3%	9.1%	9.1%
	% unique authored annotations viewed by at least one participant	23.0%	60.9%	—*
Finding: *Most participants used annotations to reveal further information about elements of the code examples, but only for a minority of the documents they looked at.*				
* We did not aggregate the coverage of unique annotation over the two sections as the documents changed slightly between the sections.				
2.3	*Split Information into Small Fragments*			
	Annotation views	356	1889	2245
	… viewed by only hovering on the anchor	311	1632	1943
	… viewed by clicking on the anchor	43	227	270
	… viewed without interacting with the anchor	2	30	32
	Authored annotation viewed from the anchor	228	1385	1613
	… with a nested anchor	41	131	172
	… with an anchor in the code example	187	1254	1441
Finding: *Participants mostly viewed annotations in floating boxes as opposed to pinned dialogs. Participants viewed nested annotations at a relative rate similar to top level annotations.*				
2.4	*Structure Information with Explicit Hints*			
	Breadcrumbs used	0	1	1
	Undo/redo buttons used	0	1	1
	Search queries	4	208	212
	… where the participant hovered over the results without selecting one	0	32	32
	… where the participant selected at least one result	2	20	22
	Participants who used the search bar at least once	3	43	46
	Document views with at least one search query	3	159	162
	% inline (vs block) markers seen by participants (average)	57.8%	61.1%	60.3%
	% inline (vs block) markers interacted with by participants (average)	85.6%	86.9%	86.6%
Finding: *Participants did not rely often on secondary navigation aids. They interacted with block markers less often than with inline markers, which are closer to the anchor and have a higher color contrast.*				
2.5	*Support the Integration of External Content*			
	Unique annotations in all documents	1565	1529	—
	… with only Javadoc content	1148	1112	—
	Annotation views	356	1889	2245
	… with only Javadoc content	128	485	613
Finding: *API reference documentation augmented code examples with a considerable number of annotations without additional effort. These imported annotations were used by participants, representing a quarter to over a third of all annotation views.*				

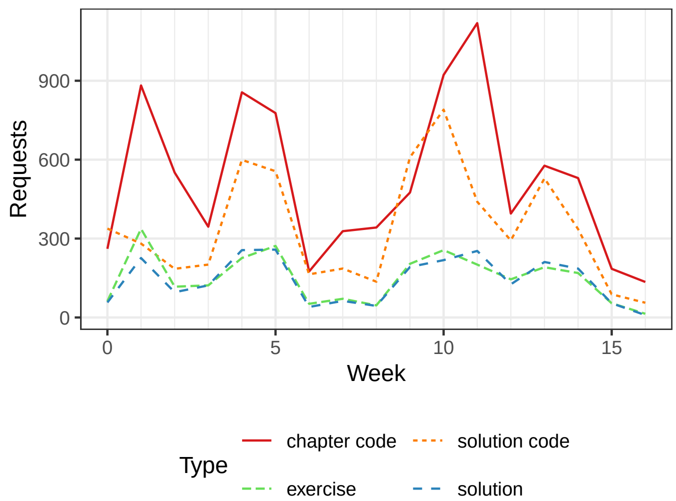


Fig. 5. Requests to Each Type of Document Received by the Study Website During Each Week of Section 2

of requests for code examples to requests for other documents. Figure 5 shows the number of weekly requests for each document type. Users looked at code examples, in particular chapter code, almost three times as often as exercise and solution descriptions. The number of requests fluctuated over time, but the usage of code examples was strongly correlated to the exercise and solution descriptions (Pearson's $r = 0.93$ between the distributions of weekly requests).[24]

Comparing the solution code and description requests can provide more detailed insights into the usage of text-oriented and code-oriented documents. There were nine solution descriptions (i.e., one document per chapter), each linking to an average of 3.8 solution code examples. Each solution description document presents the complete solution to all exercises, with smaller code fragments to illustrate the main part of a solution. However, we still observed that, for each chapter, users requested solution code examples 2.4 times more often than solution descriptions on average. This ratio did not vary considerably per chapter, even for the two chapters where the solution descriptions did not include any link to solution code (ratios of 2.1 and 1.7). For each chapter, the ratio of solution code to solution description requests ranged from 0.9 and 1.5 (first two chapters) to 4.1 and 3.0 (chapters four and five, for which the description had the most links). *[HYP] The consistent use of code examples demonstrates the value of this format to complement concise textual descriptions.*

RQ2.2. Reveal Information Gradually. Participants did not often reveal the content of annotations in a code example. Although 115 participants (76.7%) used annotations at least once, they opened one or more annotations in only 15.6% of the code examples they consulted (excluding code examples that did not contain any annotation). Furthermore, participants opened only 9.1% of the annotations with a visible marker in the code example, i.e., annotations that are not nested inside others or Javadoc annotations. Thus, participants used annotations, a key feature of Casdoc, only sporadically to reveal additional information about the code example. *[HYP] The low usage of annotations may reflect the purpose of dynamic dialogs in current documents, i.e., to offer optional supporting information, as opposed to serving as the primary means to organize content. Readers used to static documents may need to develop new habits and strategies to*

[24] Additionally, we did not find evidence that usage of code examples decreased significantly over time (Kendall's $\tau = -0.242$, $p = 0.175$ [36]).

reduce the cognitive load of navigating the implicit structure of dynamic documents. Another, possibly complementary explanation for the low usage rate of annotations is that the unseen annotations were not relevant to the readers. In this case, Casdoc was successful in avoiding distraction to participants by hiding irrelevant information from them.

Nevertheless, participants collectively viewed 60.9% of all authored annotations, including nested annotations. Thus, although individual participants saw only a small portion of the documentation content, the majority of the content was accessed by the audience as a group.

Looking at each participant individually, we observed some differences in their behavior. In particular, some participants used annotations much more than the average. For example, five participants consulted five or more annotations on at least 10% of the code examples they looked at (excluding code examples with no annotation at all), and three participants interacted with more than half of all the visible markers anchored in the code examples. *[HYP] This observation reinforces the hypothesis that Casdoc can adapt the content of a document to individual readers.*

RQ2.3. Split Information into Small Fragments. Fragmented information requires readers to gather relevant content from multiple places in a document. When designing Casdoc, we expected that floating annotations would favor a reading style where the document's content keeps changing, whereas pinned annotations would favor a reading style where the reader progressively organizes the content into a stable structure that suits them. During the field study, we observed a considerable imbalance between the two interaction modes, with participants viewing annotations in their floating form most of the time (87.8% of annotation views not triggered by a navigation tool). *[HYP] As pinned annotations require more user actions to manage (e.g., to open, move, and resize dialogs), future work is needed to understand whether their lower usage relative to floating annotations is due to this higher effort or if it reflects the preferred reading style of a majority of participants.*

Nested annotations may impose an additional cognitive load on readers, as not all information is accessible from the initial view of a document. We observed that the ratio of nested to top-level annotation views was 0.095, which is comparable to the ratio of code example markers participants interacted with.[25] *[HYP] Thus, we did not find evidence that the navigation effort increases with the depth of nested annotations, as participants were as likely to interact with visible top-level anchors than with nested ones.*

RQ2.4. Structure Information with Explicit Hints. Participants rarely used the secondary navigation aids. We observed only one instance of a participant using the breadcrumbs and the undo and redo buttons. The search bar was used more often, slightly over 200 times, by 43 participants (28.7%) in 159 unique document views (2.7%). In 20 cases (9.6%), the participant pinned at least one of the search results. In an additional 32 cases (15.4%), the participant hovered over the search results to reveal the content of the retrieved annotation, similarly to hovering over an annotation anchor.[26] *[HYP] The infrequent reliance on the search bar to find annotations may indicate limitations in its implementation. However, as it is a well-known feature, its low usage may also indicate that the explicit hints from markers effectively reduced its need when looking for information.*

We observed a notable difference in the type of markers that participants interacted with the most. Authored annotations with a block anchor in the code example were marked by a gray bracket in the left margin of the code example, whereas those with an inline anchor were marked by a blue underline. Of all markers seen by participants

[25]We consider only authored annotations in this comparison, as Javadoc annotations cannot be nested. We also exclude annotation views opened using navigation tools, as they do not discriminate between nested and top level annotations.

[26]The modest success rate is an imperfect measure, as we counted successive queries separately. Therefore, if a participant uses N queries before finding the information they need, this will be measured as a success rate of $1/N$. This was necessary as it is impossible to reliably infer whether the information sought by a participant changes between successive queries.

(excluding nested anchors, which are always inlined), 61.1% were blue underlines. Yet, the annotations that participants interacted with disproportionately had inline markers (86.9%, sign test comparing the inline anchors viewed to those interacted with, per participant: $p < 10^{-15}$). *[HYP] We suspect that this difference is due to the visual aspect of the two markers, as the types of anchor did not affect the content and usefulness of annotations.*

RQ2.5. Support the Integration of External Content. Importing the API reference documentation of standard Java types and members more than tripled the number of annotations available to participants, without requiring any effort from the documents' authors. Javadoc annotations also contributed to a notable fraction (25.7%) of the annotation views. Nevertheless, participants viewed authored annotations more often than Javadoc annotations, despite their lower number, *[HYP] suggesting a tension between adding more content from third-party sources and diluting the document-specific authored content.*

5.2 Lessons Learned

The promising results from the field study encourage us to pursue the development of Casdoc to improve the presentation of software documentation. We derived several lessons from observing how participants used the different features of Casdoc. These lessons guided the design of a new version of Casdoc, which we present in Section 5.4, as well as the ongoing development of further improvements. We present these lessons here. Although the lessons are derived from empirical evidence, more work is needed to reliably generalize them to various contexts and types of documentation.

Lesson 1: Interactivity accentuated the importance of prioritizing information. Participants only viewed a small proportion of the annotations, including annotations that were clearly indicated by a marker on the code example. Participants also viewed nested annotations, which required more complex interactions, considerably less often than top-level annotations. Even simple interaction patterns, such as hovering over a specific part of the document, may prevent a reader from finding some key information. Thus, the writer of a Casdoc document should minimize the depth of annotations that contain information relevant to the majority of readers, or use code comments instead of annotations for such information. These observations highlighted the impact of the placement of information on its discoverability, especially for interactive documents.

Lesson 2: Aesthetic decisions reduced the usefulness of some features. During the study, participants seemed to notice blue underline markers (for inline anchors) significantly more than gray brackets (for block anchors). Gray brackets were farther from the code element described by the related annotation than blue underlines, due to the code indentation. Brackets also had a low contrast with the documents' background. Although we deliberately designed all markers to be subtle markers, we did not explore options sufficiently thoroughly, which decreased the usability of a notable part of Casdoc. In particular, specific screen configurations sometimes exacerbated the contrast issue, rendering brackets barely visible to the readers.

Lesson 3: Participants did not need many secondary aids to navigate the structure of information. When developing Casdoc, we spent some effort on the design of secondary navigation aids. Apart from the search bar, participants almost never used any of these features to navigate the content of a document. The effort spent on their development produced few benefits for readers. Instead, we could have spent this effort on the primary navigation hints, e.g., by exploring more aesthetic options.

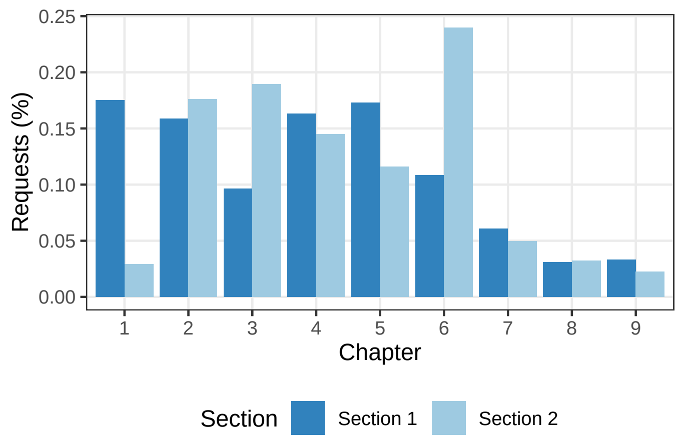


Fig. 6. Code Example Requests by Chapter from Participants of Both Sections

Lesson 4: External content had a considerable positive impact on the documents' coverage. In contrast to the secondary navigation aids, the effort spent on inserting external content reaped notable benefits. Participants benefited from over three times more annotations thanks to the integration of API reference documentation, and the usage data shows evidence that these annotations were useful. Integrating additional sources of content in the future could further improve the content of documents to address more varied needs. This must be done carefully and with the proper attribution, as the quality, style, and authoritativeness of the imported content can vary, and the external content was not originally designed for the target document. However, the effort required to integrate each external source is spent only once, regardless of the number of documents that benefit from the added content.

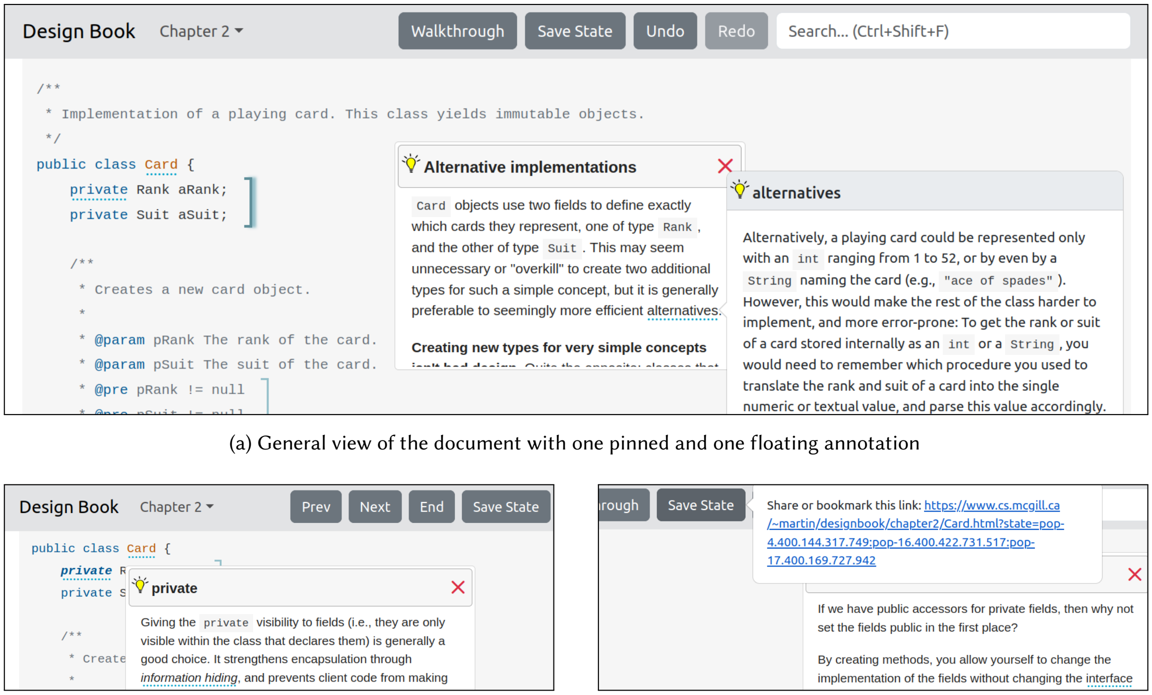
Lesson 5: Writing information as interactive annotations reduced some authoring costs. When preparing the documents for the study, we found that the annotation-based format reduced the burden of optimizing the content of a document for a specific audience. We included more information to aim for a wider audience while keeping documents approachable for individual readers. This lesson contrasts with the other ones as it relates to the authoring aspect of documents, despite the focus of our study on usage aspects. We elicited this lesson from the observation that most participants indeed used annotations and did not reveal all annotations in each document, which justifies the addition of further content that may only be relevant to some readers.

5.3 Differences Between Sections

Although the conditions of the study were less optimal during the first section, collecting data from two independent groups of participants gave us hindsight to support a triangulation of the patterns that we observed. Thus, we compared the results from the two sections of the course to identify potential divergent patterns.

Figure 6 shows the relative number of requests per chapter for each section to contextualize the collected data.[27] The number of requests decreases more consistently during the first section as chapters progress—likely a symptom of

[27] We use relative rather than absolute frequencies in this graph for better readability, as the total number of requests was considerably higher during the second section.



(a) General view of the document with one pinned and one floating annotation

(b) Navigation using the Walkthrough feature

(c) Creation of a link to save and share pinned annotations

Fig. 7. Example of a Document Using the Revised Version of Casdoc

the lower retention rate in the first section. However, the effect size remains moderate (Pearson's χ^2 test: $p < 10^{-15}$; Cramer's $V = 0.25$).

Regarding the usage metrics, we observed during the first section patterns similar to those described in Section 5.1 for most of Casdoc's features.

Among the differences we observed, participants of the first section collectively viewed a smaller proportion of all annotations than in the second section (23.0% vs. 60.9%). Individually, participants who used annotations also viewed on average fewer annotations in the first section (10.2 vs. 16.4 annotation views per participant). Thus, the majority of annotations were unused during the first section.

Despite the lower usage of annotations, participants viewed a larger proportion of annotations with nested anchors during the first section (18.0% vs. 9.5%). *[HYP] This observation may indicate that nested annotations were more useful than what we found in the main analysis, at least for some populations, but future work with a larger sample size is needed to conclusively study this hypothesis.*

Finally, although participants did not frequently use the search bar during the second section (208 usages), they almost never used it during the first section (4 usages). In particular, the lower usage rate is not proportional to the difference in the number of participants or the number of Casdoc document views. *[HYP] This observation further reinforces our interpretation that an intuitive navigation structure reduces the need to optimize secondary search features.*

5.4 Improving the Casdoc Format

Following the field study, we released the annotated code examples, without the JavaScript data collection functions, on a permanent website for students enrolled in future sections of the undergraduate course.[28] We leveraged the study findings to implement a new version of Casdoc that addresses some of its original shortcomings. Figure 7 shows one of the documents in the revised format.

According to Lesson 2, we revised the visual elements of Casdoc, taking into consideration their impact on readers. Instead of its original arbitrary color scheme, Casdoc now uses a color scheme for code blocks that is similar to the one from Stack Overflow, a popular forum among programmers (see Figure 7a). This color scheme should be more familiar to programmers, mitigating the adoption cost of a new format. To make block anchors stand out more, their markers appear at the right of the code and are blue instead of light gray. This increases the contrast of block markers with the background and reduces the distance between the marker and indented code. Finally, anchors of pinned annotations no longer show a "pin" icon next to its marker, as it was disrupting the layout of the code.[29] Instead, inline anchors are shown in bold and italics font and block anchors have their marker in darker blue and with a drop shadow to indicate that the associated annotation is pinned.

We also added three new features to Casdoc. First, authors can now identify a sequence of important annotations when creating a Casdoc document. In addition to their normal behavior, readers can reveal the annotations in such a sequence by clicking on a "Walkthrough" button (see the top of Figure 7a). The walkthrough reveals one annotation at a time, controlled by a pair of "Prev" and "Next" buttons (see Figure 7b). As they move through the sequence, readers can interact with any other annotation. The walkthrough feature was informed by Lesson 1: Although walkthroughs still require some user actions, they allow readers to access important annotations in a standard way across documents, thus reducing the cognitive effort associated with these actions. Consistently with Lesson 3, this feature also provides an additional linear structure to help readers who do not know what information to look for navigate the non-linear graph of annotations.

Second, after pinning annotations and laying them out on the page, readers can save the state of the document by clicking on a "Save State" button. Casdoc will generate a custom URL that will reopen the document with all pinned annotations already visible and in the same position (see Figure 7c). Readers can bookmark this link to keep track of annotations they find relevant. The author of a document can also use this feature to manipulate the annotations initially visible to readers. Thus, it can help make important annotations more prominent without requiring any user action (Lesson 1).

Finally, authors can now store reusable annotations in a database, and insert them in multiple Casdoc documents. In addition to mitigating the need to copy the content of recurrent annotations (e.g., an annotation that describes a common theoretical concept), the database provides a flexible interface to integrate external content into Casdoc documents (Lesson 4). The database stores the exact content of each annotation in an HTML file, associated with another file that contains properties of the annotation, such as its title. Thus, the database can be populated with external content using simple scripts, but the author of a document remains in control of which annotations are added to the document.

[28] https://www.cs.mcgill.ca/~martin/designbook/
[29] The icon was also interfering with copy-and-paste behavior, as it would be included in the copied code.

6 RELATED WORK

The importance of documentation in software development and deployment activities motivated a considerable effort to increase the usefulness and quality of documents. During an observational study, Maalej et al. found that developers prefer to look at source code or ask their peers over consulting documentation, but attributed this preference to recurrent documentation issues such as sparsity and a lack of trustworthiness [45]. Knowledge elements such as rationale, intended usage, and real usage scenarios were also often missing from documents. To improve software documentation practices, researchers studied many aspects of document generation and program comprehension, such as the types of questions asked on public forums [10, 40, 63], the information needs of programmers [11, 23, 29, 65], the overlap between the content of documents and those needs [18], and the types of information provided by different documents [2, 5, 44, 58]. Others have investigated the strategies used by programmers to navigate documents [37–39, 62]. We also discussed additional work related to the generation and usage of documentation in Sections 1 and 3.

In comparison, there is much less work on how to present information. Prior work, such as that of Zhang et al. [83], Subramanian et al. [68], and Aghajani et al. [1], include techniques to annotate code examples with information from external sources. However, their contributions are mainly about the benefits of the added content, and the automated approach to retrieve it. In contrast, our research focuses on the presentation aspects of documents and their impact on readers.

Over 30 years ago, Curtis et al. studied different strategies for presenting information about the control flow of small programs [19]. They found that using a constrained language was typically more effective than natural language or ideograms, but the arrangement of the content (i.e., whether it is shown sequentially, hierarchically, or using branches) did not have a considerable impact. More recently, Ernst and Robillard studied the format of architectural documents [24]. Their findings are consistent with Curtis et al.'s, but take into consideration modern formatting guidelines [17]. Many researchers focused on new documentation media, in particular video tutorials, as they became popular alternatives to text-based documents [16, 50, 75].

More fundamental work can help build theories to better understand, measure, and predict the quality of software documentation. For example, using neuroimaging, Sharafi et al. found similarities between programming tasks, such as understanding data structure manipulations, to seemingly unrelated mental exercises such as 3D spatial rotations [64]. Hu et al. founds limitations in how well automated documentation quality metrics (e.g., ROUGE, BLEU, METEOR) can replace human judgment along six quality factors (e.g., naturalness, understandability) [34]. This line of research is crucial to systematically improve documentation practices, but it is currently insufficient to provide concrete guidance to document authors.

Nevertheless, there has been a resurgence of effort in the human–computer interaction (HCI) field to explore new strategies when presenting text-based online digital documents. Badam et al. designed Elastic Documents, a system to dynamically link text that describes data to corresponding elements in tables and charts [7]. Similarly, Charagraph can dynamically generate data visualizations from a static document [47]. Both studies found that participants answered more accurately comprehension questions when using their system instead of a static viewer. Besides data visualization, Dragicevic et al. discussed the creation of interactive scientific articles that embed multiple data analyses, to allow readers to explore alternatives to what the researchers initially present [22]. Head et al. studied the use of techniques, including interactive ones, to help readers understand mathematical equations in documents [30]. Both of these studies focus on the design of the new format. Among other findings, they provide guidelines for creating, or facilitating the creation of, improved document formats. Finally, Symphony, a framework to create interactive documents that describe machine learning models, allows readers to see the impact of different configurations for a model using various visualizations [8]. Three case studies showed that it encouraged the debugging and validation of data sets, supported learning activities, and was an effective means of communication in cross-functional teams.

Specifically for software documentation, Codelets are self-documenting code fragments stored in a database [55]. A Codelet combines both the (potentially interactive) template code to implement a specific programming task and additional explanations. In a laboratory experiment, Codelets helped participants create a website faster than a control group. This approach is an interesting counterpart to Casdoc: Casdoc's annotations form a cohesive set of information fragments related to the document's purpose. In contrast, Codelets are reusable independent fragments that programmers authors and programmers share and integrate directly in their code. Another closely related approach is Adamite, a browser extension that allows readers to create and share their own annotations on static web pages [33]. This approach complements Casdoc by studying the benefits of user-created annotations, which can provide a new perspective on the content created by the original author of the document.

With Casdoc, our goal was to continue this exploration of alternative documentation formats by proposing an interactive, non-linear organization of information across a document. Instead of a laboratory experiment, we conducted a field study over several months to understand how readers interact with the documents in a natural setting, without the pressure to accomplish specific tasks. This difference in methodology could explain some of the variations in our findings. For example, we found that requiring readers to interact with a document can hide important information, rather than encourage an active reading, as Badam et al. observed [7]. Nevertheless, both our study and prior work show evidence that readers can effectively engage with dynamic document formats, which motivates future work in this area.

7 CONCLUSION

Motivated by the difficulty of creating concise and easily navigable documents that address the needs of a large audience, we designed Casdoc, a novel presentation format for software documents. Casdoc is intended for learning resources that focus on the completion of programming tasks, such as tutorials and usage examples for application programming interfaces. Each document centers around a compilable code example, to which are attached textual annotations that explain its different elements. Readers dynamically reveal and discard annotations by interacting with the elements the annotation describes. As annotations can also be nested within each other, they form a graph that readers can navigate based on specific needs.

We evaluated Casdoc in a field experiment with 326 participants who used over 100 documents during several months. The study focused on the impact of Casdoc's features on the participants' behavior when navigating the content of a document. Although they had access to a baseline format that contained the same information, participants overwhelmingly chose Casdoc as their preferred format. The data collected from their interaction with Casdoc documents allowed us to assess the impact of five documentation format properties on the information that readers consume. We learned from our observations five lessons that we applied to improve the design of Casdoc.

Consistently with prior work, our results show that readers appreciate interactive formats for text-based learning resources. However, they also highlight the challenges of creating effective formats. For example, visual elements and the inherent understandability of structural hints should be carefully assessed when designing an interactive format, as they may bias the information that readers are more likely to find. By striving to address these challenges, we aim to increase the overall quality of the documentation landscape for software technologies.

ACKNOWLEDGMENTS

We are grateful to Zara Horlacher and Emily Shannon for their contribution to the implementation of Casdoc. We also thank the anonymous study participants, our colleagues, and the reviewers and conference attendees for their valuable feedback on different versions of Casdoc. In particular, we thank the associate editor and anonymous reviewers of TOSEM for their keen insights that helped us improve the quality of this article. This work was supported by the Natural Sciences and Engineering Research Council of Canada and the Fonds de Recherche du Québec – Nature et technologies.

REFERENCES

[1] Emad Aghajani, Gabriele Bavota, Mario Linares-Vásquez, and Michele Lanza. 2021. Automated Documentation of Android Apps. *IEEE Transactions on Software Engineering* 47, 1 (2021), 204–220.

[2] Emad Aghajani, Csaba Nagy, Mario Linares-Vásquez, Laura Moreno, Gabriele Bavota, Michele Lanza, and David C. Shepherd. 2020. Software Documentation: The Practitioners' Perspective. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering*. 590–601.

[3] Emad Aghajani, Csaba Nagy, Olga Lucero Vega-Márquez, Mario Linares-Vásquez, Laura Moreno, Gabriele Bavota, and Michele Lanza. 2019. Software Documentation Issues Unveiled. In *Proceedings of the IEEE/ACM 41st International Conference on Software Engineering*. 1199–1210.

[4] Pavlo D. Antonenko and Dale S. Niederhauser. 2010. The influence of leads on cognitive load and learning in a hypertext environment. *Computers in Human Behavior* 26, 2 (2010), 140–150.

[5] Deeksha M. Arya, Mathieu Nassif, and Martin P. Robillard. 2020. A Data-Centric Study of Software Tutorial Design. *IEEE Software* 39, 3 (2020), 106–115.

[6] T. K. Attwood, D. B. Kell, P. McDermott, J. Marsh, S. R. Pettifer, and D. Thorne. 2010. Utopia documents: linking scholarly literature with research data. *Bioinformatics* 26, 18 (2010), i568–i574.

[7] Sriram Karthik Badam, Zhicheng Liu, and Niklas Elmqvist. 2019. Elastic Documents: Coupling Text and Tables through Contextual Visualizations for Enhanced Document Reading. *IEEE Transactions on Visualization and Computer Graphics* 25, 1 (2019), 661–671.

[8] Alex Bäuerle, Ángel Alexander Cabrera, Fred Hohman, Megan Maher, David Koski, Xavier Suau, Titus Barik, and Dominik Moritz. 2022. Symphony: Composing Interactive Interfaces for Machine Learning. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*. Article 210, 14 pages.

[9] David Beazley and Brian K. Jones. 2013. *Python Cookbook: Recipes for Mastering Python 3* (3rd ed.). O'Reilly Media.

[10] Stefanie Beyer, Christian Macho, Massimiliano Di Penta, and Martin Pinzger. 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 (2020), 2258–2301.

[11] Abir Bouraffa and Walid Maalej. 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*. 71–77.

[12] Joel Brandt, Philip J. Guo, Joel Lewenstein, Mira Dontcheva, and Scott R. Klemmer. 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*. 1589–1598.

[13] Raymond P. L. Buse and Westley Weimer. 2012. Synthesizing API Usage Examples. In *Proceedings of the 34th International Conference on Software Engineering*. 782–792.

[14] John M. Carroll, Penny L. Smith-Kerker, James R. Ford, and Sandra A. Mazur-Rimetz. 1987. The Minimal Manual. *Human-Computer Interaction* 3, 2 (1987), 123–153.

[15] Xiaofan Chen and John Grundy. 2011. Improving Automated Documentation to Code Traceability by Combining Retrieval Techniques. In *Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering*. 223–232.

[16] Pei-Yu Chi, Sally Ahn, Amanda Ren, Mira Dontcheva, Wilmot Li, and Björn Hartmann. 2012. MixT: Automatic Generation of Step-ty-Step Mixed Media Tutorials. In *Proceedings of the 25th annual ACM symposium on User interface software and technology*. 93–102.

[17] Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Paulo Merson, Nord Robert, and Judith Stafford. 2010. *Documenting Software Architectures: Views and Beyond* (2 ed.). Addison-Wesley Professional.

[18] Filipe Roseiro Cogo, Xin Xia, and E. Hassan, Ahmed. 2023. Assessing the Alignment between the Information Needs of Developers and the Documentation of Programming Languages: A Case Study on Rust. *ACM Transactions on Software Engineering and Methodology* 32, 2, Article 43 (2023), 48 pages.

[19] Bill Curtis, Sylvia B. Sheppard, Elizabeth Kruesi-Bailey, John Bailey, and Deborah A. Boehm-Davis. 1989. Experimental Evaluation of Software Documentation Formats. *Journal of Systems and Software* 9, 2 (1989), 167–207.

[20] Rodrigo Fernandes Gomes da Silva, Chanchal K. Roy, Mohammad Masudur Rahman, Kevin A. Schneider, Klérisson Paixão, Carlos Eduardo de Carvalho Dantas, and Marcelo de Almeida Maia. 2020. CROKAGE: effective solution recommendation for programming tasks by leveraging crowd knowledge. *Empirical Software Engineering* 25, 6 (2020), 4707–4758.

[21] Diana DeStefano and Jo-Anne LeFevre. 2007. Cognitive load in hypertext reading: A review. *Computers in Human Behavior* 23, 3 (2007), 1616–1641.

[22] Pierre Dragicevic, Yvonne Jansen, Abhraneel Sarma, Matthew Kay, and Fanny Chevalier. 2019. Increasing the Transparency of Research Papers with Explorable Multiverse Analyses. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*. Article 65, 15 pages.

[23] Ekwa Duala-Ekoko and Martin P. Robillard. 2012. Asking and Answering Questions About Unfamiliar APIs: An Exploratory Study. In *Proceedings of the 34th IEEE/ACM International Conference on Software Engineering*. 266–276.

[24] Neil A. Ernst and Martin P. Robillard. 2023. A study of documentation for software architecture. *Empirical Software Engineering* 28, 5, Article 122 (2023), 23 pages.

[25] Zhipeng Gao, Xin Xia, David Lo, John Grundy, Xindong Zhang, and Zhenchang Xing. 2023. I Know What You Are Searching for: Code Snippet Recommendation from Stack Overflow Posts. *ACM Transactions on Software Engineering and Methodology* 32, 3, Article 80 (2023), 42 pages.

[26] R. Stuart Geiger, Nelle Varoquaux, Charlotte Mazel-Cabasse, and Chris Holdgraf. 2018. The Types, Roles, and Practices of Documentation in Data Analytics Open Source Software Libraries. *Computer Supported Cooperative Work* 27, 3-6 (2018), 767–802.

[27] John Gruber. 2004. *Daring Fireball: Markdown*. Retrieved 2023-08-11 from https://daringfireball.net/projects/markdown/

[28] Andrew Head, Jason Jiang, James Smith, Marti A. Hearst, and Björn Hartmann. 2020. Composing Flexibly-Organized Step-by-Step Tutorials from Linked Source Code, Snippets, and Outputs. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*. Article 669, 12 pages.

[29] Andrew Head, Caitlin Sadowski, Emerson Murphy-Hill, and Andrea Knight. 2018. When Not to Comment: Questions and Tradeoffs with API Documentation for C++ Projects. In *Proceedings of the ACM/IEEE 40th International Conference on Software Engineering*. 643–653.

[30] Andrew Head, Amber Xie, and Marti A. Hearst. 2022. Math Augmentation: How Authors Enhance the Readability of Formulas using Novel Visual Design Practices. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*. Article 491, 18 pages.

[31] Fred Hohman, Matthew Conlen, Jeffrey Heer, and Duen Horng (Polo) Chau. 2020. Communicating with Interactive Articles. *Distill* 5, 9, Article e28 (2020). https://distill.pub/2020/communicating-with-interactive-articles

[32] Kasper Hornbæk and Erik Frøkjær. 2003. Reading Patterns and Usability in Visualizations of Electronic Documents. *ACM Transactions on Computer-Human Interaction* 10, 2 (2003), 119–149.

[33] Amber Horvath, Michael Xieyang Liu, River Hendriksen, Connor Shannon, Emma Paterson, Kazi Jawad, Andrew Macvean, and Brad A Myers. 2022. Understanding How Programmers Can Use Annotations on Documentation. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*. Article 69, 16 pages.

[34] Xing Hu, Qiuyuan Chen, Haoye Wang, Xin Xia, David Lo, and Thomas Zimmermann. 2022. Correlating Automated and Human Evaluation of Code Documentation Generation Quality. *ACM Transactions on Software Engineering and Methodology* 31, 4, Article 63 (2022), 28 pages.

[35] Qiao Huang, Xin Xia, Zhenchang Xing, David Lo, and Xinyu Wang. 2018. API Method Recommendation without Worrying about the Task-API Knowledge Gap. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*. 293–304.

[36] Maurice G Kendall. 1938. A new measure of rank correlation. *Biometrika* 30, 1/2 (1938), 81–93.

[37] Amy J. Ko, Brad A. Myers, Michael J. Coblenz, and Htet Htet Aung. 2006. An Exploratory Study of How Developers Seek, Relate, and Collect Relevant Information during Software Maintenance Tasks. *IEEE Transactions on Software Engineering* 32, 12 (2006), 971–987.

[38] Amy J. Ko and Bob Uttl. 2003. Individual Differences in Program Comprehension Strategies in Unfamiliar Programming Systems. In *Proceedings of the 11th IEEE International Workshop on Program Comprehension*. 175–184.

[39] Jiakun Liu, Sebastian Baltes, Christoph Treude, David Lo, Yun Zhang, and Xin Xia. 2021. 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*. 919–931.

[40] Mingwei Liu, Xin Peng, Andrian Marcus, Shuangshuang Xing, Christoph Treude, and Chengyuan Zhao. 2021. API-Related Developer Information Needs in Stack Overflow. *IEEE Transactions on Software Engineering* 48, 11 (2021), 4485–4500.

[41] Mingwei Liu, Xin Peng, Andrian Marcus, Zhenchang Xing, Wenkai Xie, Shuangshuang Xing, and Yang Liu. 2019. Generating Query-Specific Class API Summaries. In *Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 120–130.

[42] J.D. Long and Paul Teetor. 2019. *R Cookbook: Proven Recipes for Data Analysis, Statistics & Graphics* (2nd ed.). O'Reilly Media.

[43] Lori Lorigo, Bing Pan, Helene Hembrooke, Thorsten Joachims, Laura Granka, and Geri Gay. 2006. The influence of task and gender on search and evaluation behavior using Google. *Information Processing & Management* 42, 4 (2006), 1123–1131.

[44] Walid Maalej and Martin P. Robillard. 2013. Patterns of Knowledge in API Reference Documentation. *IEEE Transactions on Software Engineering* 39, 9 (2013), 1264–1282.

[45] Walid Maalej, Rebecca Tiarks, Tobias Roehm, and Rainer Koschke. 2014. On the Comprehension of Program Comprehension. *ACM Transactions on Software Engineering and Methodology* 23, 4, Article 31 (2014), 37 pages.

[46] Laura MacLeod, Andreas Bergen, and Margaret-Anne Storey. 2017. Documenting and sharing software knowledge using screencasts. *Empirical Software Engineering* 22, 3 (2017), 1478–1507.

[47] Damien Masson, Sylvain Malacria, Géry Casiez, and Daniel Vogel. 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*. Article 146, 18 pages.

[48] Damien Masson, Sylvain Malacria, Edward Lank, and Géry Casiez. 2020. Chameleon: Bringing Interactivity to Static Digital Documents. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*. Article 432, 13 pages.

[49] Brad Miller and David Ranum. 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*. 150–155.

[50] Parisa Moslehi, Juergen Rilling, and Bram Adams. 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, Article 111144 (2022), 21 pages.

[51] Seyed Mehdi Nasehi, Jonathan Sillito, Frank Maurer, and Chris Burns. 2012. What Makes a Good Code Example? A Study of Programming Q&A in StackOverflow. In *Proceedings of the 28th IEEE International Conference on Software Maintenance*. 25–34.

[52] Mathieu Nassif, Alexa Hernandez, Ashvitha Sridharan, and Martin P. Robillard. 2022. Generating Unit Tests for Documentation. *IEEE Transactions on Software Engineering* 48, 9 (2022), 3268–3279.

[53] Mathieu Nassif, Zara Horlacher, and Martin P. Robillard. 2022. Casdoc: Unobtrusive Explanations in Code Examples. In *Proceedings of the 30th IEEE/ACM International Conference on Program Comprehension*. 631–635.

[54] Mathieu Nassif and Martin P. Robillard. 2023. A Field Study of Developer Documentation Format. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*. Article 7, 7 pages.

[55] Stephen Oney and Joel Brandt. 2012. Codelets: Linking Interactive Documentation and Example Code in the Editor. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 2697–2706.

[56] Peter Pirolli and Stuart Card. 1999. Information Foraging. *Psychological Review* 106, 4 (1999), 643–675.

[57] Luca Ponzanelli, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, and Michele Lanza. 2014. Prompter: A Self-Confident Recommender System. In *Proceedings of the IEEE International Conference on Software Maintenance and Evolution*. 577–580.

[58] Daniele Procida. 2017. *Diátaxis documentation framework*. Retrieved 2023-08-11 from https://diataxis.fr/

[59] Anastasia Reinhardt, Tianyi Zhang, Mihir Mathur, and Miryung Kim. 2018. Augmenting Stack Overflow with API Usage Patterns Mined from GitHub. In *Proceedings of the 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 880–883.

[60] Martin P. Robillard. 2022. *Introduction to Software Design with Java* (2 ed.). Springer.

[61] Martin P. Robillard and Robert DeLine. 2011. A field study of API learning obstacles. *Empirical Software Engineering* 16, 6 (2011), 703–732.

[62] Martin P. Robillard and Christoph Treude. 2020. Understanding Wikipedia as a Resource for Opportunistic Learning of Computing Concepts. In *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*. 72–78.

[63] Christoffer Rosen and Emad Shihab. 2016. What are mobile developers asking about? A large scale study using stack overflow. *Empirical Software Engineering* 21, 3 (2016), 1192–1223.

[64] Zohreh Sharafi, Yu Huang, Kevin Leach, and Westley Weimer. 2021. Toward an Objective Measure of Developers' Cognitive Activities. *ACM Transactions on Software Engineering and Methodology* 30, 3, Article 30 (2021), 40 pages.

[65] Jonathan Sillito, Gail C. Murphy, and Kris De Volder. 2008. Asking and Answering Questions during a Programming Change Task. *IEEE Transactions on Software Engineering* 34, 4 (2008), 434–451.

[66] Sarah Sterman, Molly Jane Nicholas, Janaki Vivrekar, Jessica R. Mindel, and Eric Paulos. 2023. Kaleidoscope: A Reflective Documentation Tool for a User Interface Design Course. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*. Article 702, 19 pages.

[67] Klaas-Jan Stol and Brian Fitzgerald. 2018. The ABC of Software Engineering Research. *ACM Transactions on Software Engineering and Methodology* 27, 3, Article 11 (2018), 51 pages.

[68] Siddharth Subramanian, Laura Inozemtseva, and Reid Holmes. 2014. Live API Documentation. In *Proceedings of the 36th IEEE/ACM International Conference on Software Engineering*. 643–652.

[69] Jiamou Sun, Zhenchang Xing, Rui Chu, Heilai Bai, Jinshui Wang, and Xin Peng. 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*. 257–268.

[70] Craig S. Tashman and W. Keith Edwards. 2011. LiquidText: A Flexible, Multitouch Environment to Support Active Reading. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 3285–3294.

[71] Jaime Teevan, Christine Alvarado, Mark S. Ackerman, and David R. Karger. 2004. The Perfect Search Engine Is Not Enough: A Study of Orienteering Behavior in Directed Search. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 415–422.

[72] Christoph Treude and Martin P. Robillard. 2016. Augmenting API Documentation with Insights from Stack Overflow. In *Proceedings of the 38th ACM/IEEE International Conference on Software Engineering*. 392–403.

[73] Christoph Treude, Martin P. Robillard, and Barthélémy Dagenais. 2015. Extracting Development Tasks to Navigate Software Documentation. *IEEE Transactions on Software Engineering* 41, 6 (2015), 565–581.

[74] Gias Uddin and Martin P. Robillard. 2015. How API Documentation Fails. *IEEE Software* 32, 4 (2015), 68–75.

[75] Hans van der Meij and Jan van der Meij. 2014. A comparison of paper-based and video tutorials for software learning. *Computers & Education* 78 (2014), 150–159.

[76] Bret Victor. 2011. *Explorable Explanations*. Retrieved 2023-08-11 from http://worrydream.com/ExplorableExplanations/

[77] Bret Victor. 2012. *Learnable Programming*. Retrieved 2023-08-11 from http://worrydream.com/LearnableProgramming/

[78] Chong Wang, Xin Peng, Zhenchang Xing, Yue Zhang, Mingwei Liu, Rong Luo, and Xiujie Meng. 2023. XCoS: Explainable Code Search Based on Query Scoping and Knowledge Graph. *ACM Transactions on Software Engineering and Methodology* 32, 6, Article 140 (2023), 28 pages.

[79] David Wong-Aitken, Diana Cukierman, and Parmit K. Chilana. 2022. "It Depends on Whether or Not I'm Lucky" How Students in an Introductory Programming Course Discover, Select, and Assess the Utility of Web-Based Resources. In *Proceedings of the 27th ACM Conference on Innovation and Technology in Computer Science Education*. 512–518.

[80] Di Wu, Xiao-Yuan Jing, Hongyu Zhang, Yang Feng, Haowen Chen, Yuming Zhou, and Baowen Xu. 2023. Retrieving API Knowledge from Tutorials and Stack Overflow Based on Natural Language Queries. *ACM Transactions on Software Engineering and Methodology* 32, 5, Article 109 (2023), 36 pages.

[81] Wan-Ching Wu, Diane Kelly, and Avneesh Sud. 2014. Using Information Scent and Need for Cognition to Understand Online Search Behavior. In *Proceedings of the 37th International ACM SIGIR conference on Research & development in information retrieval*. 557–566.

[82] Haoxiang Zhang, Shaowei Wang, Tse-Hsun (Peter) Chen, and Ahmed E. Hassan. 2021. Are Comments on Stack Overflow Well Organized for Easy Retrieval by Developers? *ACM Transactions on Software Engineering and Methodology* 30, 2, Article 22 (2021), 31 pages.

[83] Tianyi Zhang, Ganesha Upadhyaya, Anastasia Reinhardt, Hridesh Rajan, and Miryung Kim. 2018. Are Code Examples on an Online Q&A Forum Reliable?: A Study of API Misuse on Stack Overflow. In *Proceedings of the 40th IEEE/ACM International Conference on Software Engineering*. 886–896.

[84] Joerg Zumbach and Maryam Mohraz. 2008. Cognitive load in hypermedia reading comprehension: Influence of text type and linearity. *Computers in Human Behavior* 24, 3 (2008), 875–887.