Augmenting Software Documentation through Crowd-sourcing Annotations with ADAMITE

In-submission

AMBER HORVATH and MICHAEL XIEYANG LIU, Carnegie Mellon University
RIVER HENDRIKSEN, University of Pittsburgh
ANDREW MACVEAN, Google
LAI WEI and BRAD MYERS, Carnegie Mellon University

Modern software development requires developers to find and effectively utilize new software libraries and APIs. In learning these new APIs, developers typically need to familiarize themselves with documentation, but documentation has many well-known issues. We have developed the Adamite browser plugin to investigate crowd-sourcing annotations as a way of mitigating these issues. Adamite provides multiple annotations types such as question-answer, to-do, issues and highlights which help the annotation authors organize their information finding. Saving the annotations helps later developers to find answers to their questions and confusions in context in the documentation. Our user studies showed that developers were able to effectively utilize annotated documentation when completing an API learning task and created significantly higher quality code (18% more accurate than code made without using Adamite), and that developers also created high-quality annotations that were helpful to themselves and could help future users.

Additional Key Words and Phrases: annotations, software engineering, application programming interfaces (APIs), documentation

1 INTRODUCTION

Application programming interfaces (APIs), including libraries, frameworks, toolkits, and software development kits (SDKs), are used by virtually all code [20]. However, there is significant evidence that APIs are often difficult to use [23], which can cause APIs to be used incorrectly, resulting in bugs and sometimes significant security problems [8, 30]. Nonetheless, programmers at all levels must continually learn and use new APIs in order to complete any project of significant size or complexity.

In learning and correctly using APIs, the documentation, including tutorials, reference documentation, and code examples, along with third party question-and-answer sites like Stack Overflow, are critical resources for developers [15]. For decades, researchers and practitioners have looked for ways to improve documentation, but users still complain about its poor quality, such as that the documentation containing ambiguous and incomplete information [1], which can severely block users [29].

In order to address some of these shortcomings of documentation, we are investigating the use of annotations to help developers who are new to an API expand upon, question, and keep track of information found in API documentation. Annotations are commonly defined as meta-level content that is anchored to a specific piece of text. A popular current kind of annotations are comments in Microsoft Word, Google Docs or Overleaf. Considering that some of the key problems of documentation are inherent to the documentation’s content, we posit that publicly shared annotations, which are anchored to the relevant places in the documentation, may help to address these shortcomings. These annotations may also be beneficial to the author since the act of annotating allows for externalization of thought which can offload some of the cognitive processes undertaken during the act of...
reading complex documentation [13]. There is also evidence that developers are already taking these kinds of notes as a way of externalizing information to assist in comprehension [17]. However, to our knowledge, there is no specific tooling support to help developers take notes and make annotations.

These annotations may then prove beneficial to other developers who have similar questions about the documentation. In contrast to social question and answer sites such as Stack Overflow, annotations provide a tight coupling between the original source of information and the commentary on it. In a study of what causes Stack Overflow questions to go unanswered, one of the largest factors was a lack of clarity in the question due to lack of context [19]. Annotations are likely to not have this issue as the context is inherent to the annotation, given the anchor point. As discussed in Robillard and DeLine’s field study of API learning obstacles, developers also have many questions about the documentation itself that are not easily answered, and annotations may serve as a way of facilitating the dialogue among users of the API, or between users and key stakeholders such as the API designers or even the documentation writers, who need concise user feedback about their documentation [22, 24]. Further, developers will do work to help documentation and some companies want mechanisms that facilitate that, such as Microsoft and their usage of the Open Publishing model [25].

Given the flexible nature of annotations, they can serve a variety of roles. A literature review by Agosti et al. of annotations in general reported that annotations are commonly used in five ways: as a comment (in which the annotator provides their own reaction to the text), elucidation (to make a point more clear), interpretation (to explain their interpretation of the text), gloss (to give a definition for an unfamiliar word), and notes (to jot down some content to serve as a reminder for future reference) [2]. We conjectured that providing different types of annotations may address some of the well-known shortcomings of API documentation, as developers commonly encounter domain specific terms and unfamiliar API-specific functions that they may not be familiar with (thus, the need for a gloss type annotation), lose track of useful information they encounter (thus, the need for notes), and suffer from ambiguous or incorrect information in the documentation (thus, the need for elucidation and interpretation). Another common use for annotations is for strictly personal purposes such as notes that are in the style of to-do action items for the author [16] which may be considered a note [2].

To begin understanding how annotations may benefit developers when learning an unfamiliar API, we ran a preliminary study using a commercially-available annotation tool, Hypothesis [12]. This study showed promise as developers were able to make use of annotations present in the documentation, but Hypothesis was not entirely satisfactory as it lacked mechanisms to help developers author and find useful annotations. We also analyzed the public annotations that programmers had made using Hypothesis on some Stack Overflow and GitHub web sites, and discovered that programmers are capable and willing to write the annotations we believe would be helpful, including explanations of poorly represented concepts, questions of the underlying content that we conjecture would have been helpful had the questions been addressed, issues with the documentation including typos and out-of-date information, and personal to-do notes for the developers.

As an initial exploration about how annotations could be made easier to author and more useful for later developers, we built our own tool, Adamite1, as shown in Figure 1. Adamite attempts to address the limitations of Hypothesis and more directly supports the needs of developers through more advanced authoring and anchoring mechanisms, private and public annotations, and supporting different types of annotations such as keeping track of to-do items and open questions to help developers better create useful annotations, and through searching and filtering mechanisms to help later developers find the most relevant annotations for their needs. In our user studies of Adamite, we found that developers were able to effectively use Adamite for authoring high-quality annotations, and other developers were significantly more accurate at completing an API learning task than those who did not use Adamite.

1Adamite is a green mineral, and here it stands for Annotated Documentation Allows for More Information Transfer between Engineers. The Adamite software is available opensource at [anonymised].
Apache Beam Programming Guide

The Apache Beam Programming Guide is the introductory documentation web-page for the Apache Beam API, with Adaitme open on the right. Adamite presents annotations in a sidebar view with their anchor locations highlighted on the page. At the top right of the sidebar, the user can click on the user icon to log out of Adamite, and click on the file icon to add a page level annotation. The next row contains the filter icon and the search bar. The current filters are summarized in the next row, and clicking on the filter icon brings up the filter menu shown in Figure 3. The user can click on the “Show Pinned Annotations” button to see all of their pinned annotations across all websites. The annotations shown in the sidebar are all of different types, with the type signified by the icon in the upper right of each annotation. The five icons below that support replying, pinning, adding additional anchors, editing and deleting this annotation. The fourth annotation is a question-type annotation that has been answered, with the answer’s content added to the main annotation body. The fifth annotation is collapsed and its anchor is highlighted in green both on the main page and in the Adamite panel to show where it is on the page, because the user’s mouse is over this annotation. Clicking on the anchor in an annotation scrolls the document to the referenced part of the web page. Annotations can be collapsed and expanded by clicking the arrow icon on the bottom of the annotation.

The contributions of this paper are:

- A preliminary user study (Section 3) and data mining analysis (Section 4) of the ways in which annotations could help programmers better utilize documentation.
- A new tool, Adamite, that is designed to address programmers’ information needs that are not being met by current documentation or annotation tools (Section 5).
Two lab studies that show the efficacy of Adamite for developers who are using annotated documentation and for developers who are creating their own annotations during an API learning task (Sections 6 and 7).

2 RELATED WORK
Understanding how developers learn unfamiliar APIs and developing tools to assist in this learning process have been the subject of much prior research. Other work has looked at the use of annotations and systems that support annotations in domains other than programming, or at issues with documentation.

2.1 Studies of Programmers Learning in General
There are many studies of programmers learning unfamiliar code, some of which are relevant to annotations of documentation. LaToza et al. [14] discussed that programmers need to learn many facts while understanding code, and would benefit from a way to record what they learned, which was one inspiration for our providing annotations as a way for programmers to do this. Another study by Duala-Ekoko et al. found that developers have many questions that they ask when learning unfamiliar APIs that are not trivial to answer by merely reading the API documentation [7]. When developers have these questions, their journey to find an answer may not be archived in any way, as they eventually find an answer and have no easy way of attaching it to the point that inspired the question. An example of this is illustrated in a study by Fournier and Ringel Morris which found that developers typically conduct around 45 minutes worth of research prior to asking or answering a question on a developer forum [10]. Further, that research is not represented in the actual question or answer that the developer ends up posting, despite the fact that prior work has reported that answers on question-answer forums that include context are more highly rated [21]. Sometimes developers get lost during their question-asking and answering process and lose track of their questions or forget the answers they found to their questions [27]. Our work aims to help developers keep track of their questions in context in the documentation, along with the useful pieces of information they find that answers their questions.

2.2 Previous Research on Annotations and Annotation Systems
As previously mentioned, a prior literature review found that the flexible nature of annotations allows them to serve a variety of purposes [2]. Other work noted that annotations may be seen as a conversational tool among the document users, as well as with the document creators [9]. Therefore, our tool supports these conversational and navigational goals by connecting useful parts of documentation through linking to where concepts are explained while supporting expansion of the original text.

Another perspective on annotations looks at whether and with whom they are shared, and how that changes their form. Prior work found that only a small percentage (around 20%) of annotations created in an English classroom setting were shared [18]. The amount of text added prior making the annotation public was also quite stark – private annotations averaged 2 to 10 words, while public annotations were 30 to 150 words. Given the amount of work that must be undertaken to move an annotation from “private” to “public”, most commercial annotation tools such as Hypothesis support annotations that are viewable only by the author (thus, private) and annotations that can be viewed by anyone (public), a feature we also support.

Other systems that support annotating on the web helped inspire and inform our design. Chilana et al.’s system LemonAid allows users to select webpage elements such as buttons and menu items and crowdsource questions about the element’s intended usage, and answers about each element [5]. Zyto et al. developed an annotation system called NB which was successful in math and science classroom but needed more complex anchoring support as sometimes participants would create annotations that were relevant in multiple places across the text [31], which our tool supports, along with global level annotations.
2.3 Studies of Documentation

Documentation, specifically API documentation, has been the subject of many research projects, often attempting to understand the particular pain-points of modern software documentation [1, 15, 24, 29], especially considering API documentation is cited as one of the most important resources but also one of the most significant obstacles when learning a new API [23]. In one survey of 323 professional developers [29], incomplete information was the most frequently cited issue with documentation that was a significant blocker to developers. Other highly referenced blockers included “ambiguous information”, “unexplained examples”, and “incorrect information”. We conjecture our tool would help with these issues through supporting annotations that can elucidate and explain ambiguous information and code examples, while also directly supporting calling out incorrect information. Conversely, we do not believe our tool would help with another issue reported in the paper – the issue of “bloat” as annotations may provide additional bloat. However, “bloat” was rated low in terms of blocking progress and was not a commonly encountered issue. Another study reported that issues with documentation led developers to explore other information sources, such as question-answer sites like Stack Overflow, blog posts, and bug reports, which can contain rich information that may be used to supplement the original API documentation [4]. For example, SISE, a machine-learning based tool, extracts information from Stack Overflow and adds this information to the original documentation [28]. However, approaches such as SISE require the existence of a large user-base of the API such that there is information available to mine, and when we attempted to apply Stack Overflow content to documentation, the length and format of these Stack Overflow excerpts made them hard for developers to use.

3 PRELIMINARY STUDY

To assess the efficacy of annotations as a useful learning device for API learning tasks and to see to what extent current annotation tools could address developers’ specific information-finding needs, we ran a preliminary study where people learned a new, unfamiliar API, Apache Beam, while using Hypothesis [12], an off-the-shelf annotation tool. The preliminary study had two distinct phases: the first phase was focused on understanding how developers author annotations during an API learning task, while the second phase focused on how developers use annotations that are already attached to a page, even if there are a large number of annotations, many of which are irrelevant to the current task.

3.1 First Phase of Preliminary Study

For the first phase of the study, participants were instructed to complete some Python code using Apache Beam (an API learning task adapted from a previous study [11]) while foraging through Beam’s documentation and other online resources for the requisite information. This task was chosen because it is difficult and requires understanding the documentation. Moreover, as discussed in [11], the Apache Beam documentation has previously-known shortcomings, such as poorly-explained concepts, a lack of code examples, and unclear terminology, which we hypothesize annotations could address, making it a good testing ground for seeing how annotations may improve the documentation. Furthermore, since Beam is relatively new and rarely used (even though it is a public API provided by a large vendor), there is little useful information in Stack Overflow, so web searches frequently do not provide useful information.

The task requires participants to use Apache Beam to complete a word counting pipeline. The participants were given a Python file which shows the 7 steps that they need to fill in to complete the program. The handling of the input, output, and running of the pipeline are already provided; the participant just needs to utilize the documentation in order to find and familiarize themselves with the Beam-specific transforms that must be called in the correct order to complete the pipeline.
Considering the study was an exploratory preliminary study, participants were recruited from the authors’ social circles. All participants were required to have some amount of experience with Python and data analysis (Beam’s primary usage context), but we were less concerned about their ability to successfully complete the task and more interested in the annotations they authored. In total, 5 participants were recruited for the first phase, but one participant had technical difficulties and could not complete the task, so we only report on 4 participants (P1-2 through P1-5). All participants were audio and video recorded and were instructed to think-aloud. Participants were given 45 minutes to complete the task and were compensated $15 for their time.

Participants were given the Beam documentation with no annotations, and instructed to add annotations when they learned anything useful, had questions about the content in the documentation, or had any other thoughts about the documentation. Participants could also annotate other websites, such as Stack Overflow, with annotations related to the task. Since we were primarily interested in the types of annotations participants naturally authored, we collected all authored annotations and looked for emergent themes.

3.1.1 First Phase Results. The 4 participants together authored a total of 21 annotations, 2 of which were duplicates, resulting in 19 unique annotations. Each participant, on average, authored 4.75 annotations, with annotations averaging 4.41 words. Hypothesis also allows users to simply highlight a piece of text on a web page without adding any text content to the anchor (hereafter referred to as “empty” annotations) — out of the 19 unique annotations authored, 5 were these simple empty annotations. Given the relatively short task and how cognitively demanding the task was, the small amount of authored annotations is somewhat unsurprising. However, when this task was repeated using Adamite, 8 developers authored 127 annotations (average of 15.9 annotations per person) (see Section 7).

When participants did create annotations they were typically to highlight a part of the documentation that illustrated how to achieve some part of their current task. For example, one participant highlighted the part of the documentation subtitled “Applying Pipeline Transforms” and annotated the phrase with the text “how to data pipeline in Beam”. These annotations are useful as a visual signpost to useful information on how to achieve some effect and are easy for users to skim as they are short in length. Other annotations served to summarize the original text or expand upon it. For example, P1-3 added an annotation on a reference to Beam’s Filter method that said “Filter can be used with a function or lambda function”, information that was not made clear in the original documentation. Such expansions are similar to Agosti’s previously documented “elucidations” and “interpretations” that serve to make information more salient and enrich the text by allowing the author to share their own knowledge [2]. Some annotations served as a task reminder or open question the author had about the content. For example, P1-3 marked a part of the example code with the annotation “TODO: need to figure out what the ‘[]’ does here”. These “to-do” style annotations are consistent with prior work about some of the shorthand notes developers make while programming [16]. In Hypothesis, there is no easy way to keep track of these annotations unless you specifically tag the annotation with some consistent tag that can be queried, but these annotations would have no additional affordances for keeping track of them or attaching the original annotation with additional information about what one learned upon completing this task. This unmet need to follow up on interesting or confusing resources directly impacted our later design of Adamite.

From this initial preliminary study, we found evidence that annotations may enhance the original text. Additionally, we found support for different types of annotations that are not directly supported by Hypothesis, such as keeping track of open questions and to-do items.

3.2 Second Phase of Preliminary Study

The second phase was an exploratory preliminary study designed to assess how developers utilized annotated documentation using Hypothesis. Participants were recruited from the authors’ social circles and all participants had some amount of experience with Python and data analysis. None of the participants in phase 2 had participated
in phase 1. In total, 5 participants (referred to as P2-1 through P2-5 in order to avoid confusion with Phase 1) were recruited and had 45 minutes to try to complete the task. All participants were audio and video recorded and instructed to think-aloud.

Participants were given the same task that was used in phase 1. We chose to keep the task the same since we could add in annotations that we felt addressed the difficulties our phase 1 participants had, along with annotations that addressed some of the issues reported in [11]. We were interested in whether participants could make use of the annotations that were designed to be useful even in the presence of many other annotations that were not likely to be helpful for the assigned task to investigate the issue of scale.

We provided all the annotations, which were of two types — annotations authored by the first author that were designed to be helpful for this task and other annotations that were designed to be “distracting”. We collected our “distractor” annotations from a number of Stack Overflow posts that were related to Apache Beam. We chose a random subset of 44 question-answer pairs that were in Python and were relatively concise and understandable. These question-answer pair annotations were anchored to the Beam Programming Guide on words or phrases that matched the text included in the original question.

In total, we had 23 “helpful” annotations and 44 “distractor” annotations, totalling 67 unique annotations. Two annotations were copied onto multiple places throughout the documentation, as their content was relevant at multiple points. Counting these annotations that appeared multiple times, the Beam programming guide had a total of 78 annotations anchored to it.

3.2.1 Second Phase Results. In order to analyze the second phase participants, we transcribed each participant’s session with the questions participants asked and the actions they undertook to try and answer these questions. We chose to analyze our data this way to understand specifically what participants were confused about, when they were confused, and where annotations may fit into their quest for an answer. In order to keep the transcription process objective, we only counted a question if the participant explicitly stated a question (e.g., P2-1 asked “How do I create a PCollection?”). Similarly, we only counted answers if the participant gave some verbal indication that they had found an answer (e.g., P2-1 answered their own question by saying “Oh, beam.Create – this is what I want”). Through this transcription process, we counted 138 questions across the 5 participants.

For the 138 questions, only 40 were definitively answered (29%). Note that, due to the objective nature of the coding, participants may have answered more questions internally but did not verbalize anything that would make it possible to tell whether the question was answered. 61% of the time, when a participant asked a question, they almost immediately followed it with a new question and worked on that one instead of answering their initial question. Many questions that went unanswered during the study and did not have an associated annotation were either too specific to the programming task to be appropriate as an annotation (e.g., asking about a function we wrote that appeared in the starter code), or were too vague to have an appropriate annotation (e.g., “Why is everything wrapped in another thing?”).

The majority of the participants’ questions occurred while they were looking at the code editor (84/138 or 61% of the time) — the rest of the times (54/138), they were looking at the documentation. When the participant attempted to address their question, they most often went to a new part of the documentation or they navigated to the documentation if they were in the editor (80/138 question instances—58%). Notably, they only went to their code 19/138 times (14%) to address their question; the rest of the times, they either navigated to the documentation, asked a new question, or navigated to another web resource such as Google or Stack Overflow. This suggests that participants typically expected the answer to the question to reside somewhere in the documentation.

Participants were slightly reluctant to interact with our annotations. Across the 5 participants, participants interacted with annotations 77 times. This was most likely influenced by our distractor annotations as since they tended to be long and irrelevant. This often caused them to lose faith in the annotations having useful information and thus valued them less as a resource. Participants tried to determine relevance based on the
annotation’s anchor. For example, when P2-1 wanted to learn how to set up their pipeline, they clicked on a distractor annotation attached to the text “pipeline options” that had a long discussion from Stack Overflow about setting up a pipeline so that Amazon Web Service information was kept private. Upon skimming the annotation, he stated the annotation was “too long!” and irrelevant and closed the Hypothesis sidebar altogether. These information mismatches led some participants to believe that none of the annotations would contain the information they really needed.

From Phase 2, we learned that annotations need to be anchored in appropriate places and easy to filter and search so that participants can quickly determine relevance and effectively utilize authored annotations. Determining an appropriate place to anchor an annotation is not trivial — users may come to an unfamiliar programming context with different preconceptions about how the API should function, resulting in differing search queries. Thus, we added support in Adamite for user-suggested anchors, where users may add a new anchor for an existing annotation at a new place in the documentation, so that the annotation is available at multiple entry-points. These findings also motivated the design of the “question” type annotations.

4 DATA ANALYSIS OF HYPOTHESIS ANNOTATIONS
In order to supplement the first phase of our preliminary study to understand what annotations developers have already made, we queried Hypothesis’s API to get a list of some public annotations developers have already made on official API documentation including for Google, Microsoft, Oracle, and Mozilla’s public APIs, along with other developer learning resources such as Stack Overflow, W3Schools, and GitHub.

Across these sites, we found 1,873 public annotations, but there may be more private annotations or annotations shared with a private group of individuals which are not accounted for in this count. We did an exploratory thematic analysis of the annotations to see whether they resembled the annotation types in prior work and the types we saw in our preliminary phase 1 study. Of the 1,873 annotations, 142 were questions about the content of the documentation, and 686 were empty type annotations where they simply marked an interesting or useful passage on the web page. Many of the remaining 1,187 annotations contained content that we believe may have the potential to benefit other developers. For example, a Hypothesis user annotated the text “Skip if possible means...” and added additional details about the behavior of the Skip method that was unclear in the original text. Other annotations of Agosti’s types [2] that we found included navigation annotations that pointed to other useful web resources, interpretations of the information (such as drawing analogies between one API’s functionality and another), and comments on the text, either as a reaction to the content itself or as a reminder to themselves in the future to complete some action. Some annotations that fit less well into Agosti’s classifications and were more developer-specific such as questions about design rationale.

These Hypothesis annotations provide support for our claim that developers are willing to write annotations and attach them to documentation, as they are already doing this. Some of the annotations were quite lengthy and thorough, with 96 annotations containing 30 or more words. Moreover, the annotations we found follow some of the previously identified patterns of authored annotations, including to-do style annotations, questions about the content, expansions upon topics, and annotations stating how a particular function works.

5 OVERVIEW OF ADAMITE
In order to address some of the developer-specific needs we discovered in our preliminary studies that are not met by current annotation tools, we developed Adamite, an annotation tool designed to help developers keep track of important information and learn from one another.
5.1 Adamite’s Design

Adamite is consistent with other annotation tools, such as Hypothesis, as well as with Google Doc comments, in allowing users to author and view annotations on the web page using a sidebar (see Figure 1). Adamite also supports searching for annotations, filtering the annotation list using a wide variety of criteria, replying to annotations, collapsing and expanding annotations, adding new anchors to annotations, and, if they are the original author of the annotation, editing and deleting annotations. Considering the sidebar may contain many annotations, only a handful of which are relevant to the user, we provide flexible filtering so users can find annotations which match their criteria. For example, users may want to see only the annotations they authored themselves (including their private annotations), or only “to-do” type annotations so they can keep track of their personal work items. Upon applying these different filter criteria, the sidebar updates with the currently viewable annotations and the count of how many there are. In addition to or instead of using filters, users can search the anchors and contents of the annotations using the search bar. Obviously, users can also find annotations by just clicking on anchors in the web page. We provide a variety of ways of finding annotations since the preliminary study showed that participants had a hard time finding useful annotations for their current problem.

Users may author their own annotations by highlighting a part of the web page, selecting the type of annotation they want to make from a menu (see Figure 2), and then typing their annotation’s content into a rich text editor in the sidebar. When authoring an annotation, users may add tags. Users may also author annotations that are anchored to the whole page through clicking a “file” button at the top of the sidebar.

5.2 Design Rationale

In designing Adamite, we wanted to make annotations easy to find and utilize given a developer’s task-specific needs, help developers author useful annotations, and minimize the interference of annotations with the documentation during API learning tasks. We frame our specific design choices around the issues that developers encountered in our studies and that have been discussed in prior literature.

- **Issue:** Typically, developers’ high level goal is not to read – it is to learn enough to continue programming. [3]
- **Design Choice:** Use a simple, easily-dismissed sidebar design.

Adamite retains the sidebar design used by other browser extensions like Hypothesis, as it is relatively unobtrusive. The sidebar supports easy dismissal when users do not need annotations (in our implementation, users may toggle a key-binding to dismiss the sidebar and can also resize the sidebar such that it does not obstruct their view of the web page). Adamite also adjusts the size for the web page content based upon the size of the sidebar so that content on the web page is not covered.

- **Issue:** Developers need help keeping track of the information that is important to them, and incomplete learning tasks. (Section 3.1)
- **Design Choice:** Support pinning annotations such that they are always accessible.

A button in the sidebar (see Figure 1) displays all pinned annotations, no matter what filter is in place. An annotation can be pinned or unpinned by clicking the pin icon. Open questions and to-do annotations are pinned by default and are automatically unpinned when answered or marked as done. We added pinning so that developers can have access to particularly helpful annotations and can also easily access information they want to follow up on, such as open questions and to-do items.

- **Issue:** Developers need more support in authoring annotations that are useful to themselves and others. (Section 3.1)
- **Design Choice:** Provide more structure to annotation authoring by supporting different types of annotations.
Fig. 2. A question-type annotation. A developer does not know the term “predicate” so they select the phrase with the unfamiliar term in the webpage (left screenshot), and use Adamite’s popup question to annotate it, which creates and pins the unanswered annotation (center). Upon finding a description of “predicate” that answers their question, they add an anchor to the answer location along with their answer summary, which automatically marks it as answered and unpins the annotation (right).

Our annotation types include questions (with answers), to-do’s, issues, normal, and empty with certain types providing additional affordances to make them useful for both the author and for later users of the documentation. We additionally support tagging of annotations such that developers can more easily organize them.

Motivated by the number of questions developers had about documentation and its content, we developed the “question” type annotation that allows developers to quickly and easily enter questions about the documentation’s content while Adamite keeps track of the state of the question. Developers can either author their own question or choose from a menu of commonly-asked questions including “what is this” and “how do I use this” (Figure 2-left). If a developer has an unanswered question about the documentation, the annotation will stay pinned until they mark the question as answered or the user marks it as no longer relevant. The answer can also contain an anchor on the information that answered their question. An answer may also be “starred” by the original author of the question which causes the answer’s content to appear in the main body of the annotation (see Figure 2).

We also support “to-do” annotations that allow developers to keep track of future actions while anchoring it to a part of the documentation that may provide additional information as to how to complete the task. To-do annotations can also be marked as “done”, and then optionally hidden, and are by default accessible in the pinned annotations menu until they are completed.

Considering a key problem with documentation is that information may be incomplete or out-of-date [29], we support “issue” type annotations that can be used to mark problematic, incomplete, or incorrect aspects of the documentation. Issue type annotations have an option to flag the annotation for expert review, which we envision could trigger a notification to the key stakeholders such as the technical writers or a developers of that API.

We also support “normal” and “empty” highlight annotations, the two annotation types supported in Hypothesis, as their flexible nature allows them to be used in a variety of ways by developers. Normal annotations may be used to expand upon the original text through “elucidations” and “interpretations” or provide some reaction as a “comment” or “note” [2]. Empty annotations allow developers to quickly highlight some part of the text they found interesting or useful without adding any additional text.

- **Issue:** Developers have a variety of background knowledge and reasons for visiting documentation so they need an easy way to find information that is relevant to their particular background and needs. (Section 3.2)
- **Design Choice:** Support searching and filtering across the whole website and by keywords or tags.
To help users find information across the large search space, we added a filter feature. With Hypothesis, there is no easy way of restricting your search space beyond changing your current web page. The Adamite filter bar, as shown in Figure 3, has multiple sections, for filtering on different properties. The filters have an OR relationship within each section and an AND relationship between filter sections. For example, an example filter would be to show annotations that are authored by myself OR other users AND are on this page, where the author of the annotation and its location are two different filter sections (this is similar to how filters work on shopping sites like Amazon or Kayak). Adamite currently supports displaying annotations that are attached to the current web page or annotations that are attached across the whole website, since documentation may contain many pages, many of which may have information that is relevant to a user’s work context. We also show a filter summary at the top of the sidebar so a developer can always see the state of their filter to reduce confusion caused by not knowing why certain search results are unavailable in their current setting, as discussed in [26].

Adamite also makes filtering using tags more useful than Hypothesis. When clicking on a tag on an annotation, the Hypothesis system opens a new browser tab which shows all annotations on every website that include that tag, making it difficult for a user to find annotations that use that tag. In contrast, Adamite combines filtering by tags with the other filters, providing a consistent and simple interface.

- **Issue**: Developers may not find useful information when it is fragmented across multiple pages of the documentation. [29]
- **Design Choice**: Support anchoring of annotations to multiple places in the documentation.

Adamite allows annotations to be attached to multiple places across multiple webpages. We added this feature as information related to programming constructs and concepts may be relevant in multiple places in the documentation. This feature also helps to address the previously noted issue of documentation information being fragmented across multiple pages and places — by allowing annotations to serve as a bridge between these disparate information locations, Adamite can assist developers in navigating between pages through hyperlinking with additional, contextual information. Since any developer may add a new anchor, this allows developers to modify already existing annotations to have anchors to where they would expect that information to be. Answers to questions may also contain anchors, so that developers can point to useful information that is relevant to the original annotation or helped them answer their question.
**Issue:** Developers need an easy way of dismissing annotations that either have content they already know or are irrelevant to their task even when those annotations are included based on the current filter. (Section 3.2)

**Design Choice:** Support collapsing and archiving annotations so that developers can focus on only the annotations they care about.

From the second phase of our preliminary study with Hypothesis, we learned that participants needed to be able to quickly skim through unhelpful annotations. Therefore, we added the feature of collapsing annotations so each one takes up only a small amount of space on the sidebar (see Figure 1). We also support archiving annotations that are no longer relevant, so they are usually not visible (but still can be brought back if needed).

## 6 ANNOTATION USAGE STUDY WITH ADAMITE

### 6.1 Method

In order to assess the usefulness of Adamite for developers using already-annotated documentation, and see if Adamite seemed to be better than Hypothesis in this regard, we ran a user study consisting of four parts. First, participants completed a survey about their programming background and proficiency, their experiences with APIs similar to Beam, their note-taking habits, and their familiarity with systems that support annotation, such as Google Docs or Microsoft Word. Next, we trained participants by giving them an annotation authoring task which was also used to assess the usability and learnability of Adamite. Participants were asked to author a new annotation, edit it, add new tags, create an annotation of a different type ("to-do"), delete the newly created annotation, and use the filtering mechanism to find an annotation anchored to a different page. This training was performed on the ReactJS documentation, so participants did not have extra time to explore Beam’s documentation.

The third part was a coding task using annotations. We reused the task from the preliminary studies, which is also identical to the task in prior work [11]. In this way, we can see whether our tool is more helpful in comparison to the baseline of no annotations (as in [11]), and compared to having annotations but with less tooling support (see Section 3.2). We utilized the same 78 annotations that were present in Section 3.2, with the distracting annotations shortened and summarized. Considering the small number of participants in our preliminary study of Hypothesis, we do not have the statistical power to assess whether Adamite is “better”, beyond our qualitative impressions.

However, we are able to compare with the prior study [11]. One difference, which would seem to favor the prior study, is that for the current study, we did not provide participants with training about Beam beyond stating the objectives of the task. In the previous study, the experimenter provided a 10 minute overview explaining Beam, PTransforms (the main pipeline logic of a Beam program), and PCollections (the main data storage object in Beam). We chose to omit this training as all of the training material provided was present in our annotations.

At the end of the study, participants completed a short survey with questions adapted from the Technology Acceptance Model (TAM) [6] and free-response questions about their experience using Adamite. We chose to adapt TAM since Adamite is still in its infancy so understanding its usability breakdowns and perceived usefulness can inform what changes need to be made for the next iteration of Adamite.

### 6.2 Participants

We recruited 11 participants from departmental mailing lists at our university and participants with programming backgrounds from the authors’ social circles. One participant could not finish the study due to technical difficulties, so we only report on the 10 who completed the whole study.

All of the participants had some amount of experience using Python, had never used Apache Beam before, and had been programming for at least 2 years (minimum: 2 years, maximum: 10 years, average: 5.75 years). The participants’ professions included masters students in computer science-related fields, undergraduates...
in computer science, one user experience researcher with a computer science background, and professional programmers. The average years of professional software development experience among all participants was 2.2 (minimum: 0 years, maximum: 9 years). 3 participants had used distributed data processing APIs before (a factor that was significant in determining task success in [11]), 5 had used other data processing APIs or data analysis APIs such as numpy or pandas, and 2 had never used any data processing or analysis APIs at all.

Due to the COVID-19 pandemic, all study sessions were completed remotely using video conferencing software. Participants were audio and video recorded, save for one participant who did not have access to software that could support screen recording, so the first author took notes during this session. The study sessions took approximately 90 minutes for each participant, with 45 minutes of that time allotted for the programming task. Participants were compensated $15 for their time.

Similarly, as reported in [11], the 10 participants in the control group were familiar with Python, had no experience with Apache Beam, and had similar programming backgrounds and years of experience.

6.3 Analysis
The annotation authoring (training) task was graded objectively. Considering there were 7 tasks they needed to complete (by authoring 3 annotations, editing 1 annotation, adding tags, deleting an annotation, and finding an annotation on a different web page), we simply graded whether or not they were able to complete each task successfully.

In order to assess the success of Adamite during the API learning task, we graded participants on how well they completed the task by grading their completed programs. We graded programs based upon how many steps they successfully completed in the 7 step data pipeline, as discussed in Section 3.1. Each step in the pipeline had a set of transforms that were required for completing that particular step, so choosing one of those appropriate transforms earned a half point. A full point was awarded for also supplying the correct arguments to the transform. No points were awarded for incorrect transforms or if the program failed to run. We chose this grading schema as it is objective so grading was straightforward. We were able to get a copy of the programs from [11] in order to grade them according to our current schema, as the previously reported grading schema [11] was based solely on whether or not a transform existed in their pipeline and not on whether the code could actually run or not.

The TAM survey responses used a scale of 1 to 7 where 1 is “strongly disagree” with the provided statement and 7 is “strongly agree”. Since each statement is positive in nature (e.g., “I consider Adamite easy to use and easy to achieve what I want”), larger numbers are better. The free-response survey responses were assessed for emergent themes and are used to provide additional context when discussing the lab study and authoring results.

6.4 Results and Discussion
6.4.1 Annotation Authoring and Training. All participant successfully created a new annotation, edited one of their annotations, and deleted an annotation. However, some of the more advanced features were a bit difficult for some participants and some interactions were not intuitive resulting in usability issues.

Creating a “to-do” type annotation was unclear to many participants (5/10 successfully created a to-do), due to unclear instructions and UI features which were subsequently fixed. The design used during the study required participants to create a new annotation through selecting text, then choosing an annotation type using a drop down menu. Now, the user selects the initial type after selecting the text on the web page (shown in Figure 2-left).

The other task with which users struggled during the training was finding an annotation through adjusting their filters (6/10 successfully completed this task). Participants were instructed to try and find an annotation that says “thank you so much”. The key is that the annotation is on a different page that is still on this website, so by adjusting their filter to show annotations across the whole site, the annotation appears in the list. However, most participants first strategy was to search “thank you so much” in the search bar and, upon that not working,
they would search the web page or open the filter but not toggle any of the filter options. As a partial fix, we added the filter summary pane to try to make more clear the scope of the search (Figure 3-left), but in the future we will explore having the textual search always be global, or automatically switch to global if local search fails.

6.4.2 API Learning Task. The 10 participants using Adamite were able to produce significantly better programs than the 10 participants in the control condition [11] when graded consistently (paired T-test, \( p = .04 \), 18% more accurate). The average grade for the Adamite participants was a 3.6 out of 7, with a minimum grade of 0.5 and a maximum grade of 6 (standard deviation: 1.776). The mode was 5 and the median 4. This stands in stark contrast to the control, as reported in [11] (where there were no annotations) where the average was 2.3, median was 2, and mode was 1.5 (see Figure 4). On the final questionnaire, 9 out of 10 participants said Adamite made the documentation more useful. Informally, the 5 participants in the preliminary Phase 2 study in Section 3.2 had an average score of 2.5, so Adamite seems to perform better than Hypothesis as well, but there are too few participants for a statistical test.

In [11], the authors reported that participants with experience using distributed data processing APIs performed significantly better on the task than those without — however, we found no significant difference between these two groups in our study (average with parallel: 4.6, average without parallel: 3.1, unpaired T-test, \( p = .16 \)). This may be because of the new grading scheme, where Horvath et al.’s study focused on discovery while our study grades programs on their correctness. Another possibility is that there was not enough participants to see a significant difference. Alternatively, the presence of Adamite may have helped users with less background knowledge more effectively utilize the documentation. For example, P5, who had only used data analyses APIs and not distributed APIs before and performed well on the task (score = 5), stated that Adamite was very helpful because “… it [was] easier to understand what was going on, especially because the annotations could fill in if the documentation was worded weirdly or missing some useful information”. Other background characteristics of programmers, such as whether or not they were a professional programmer or used Google Docs comments before were also insignificant factors in terms of task success.

The annotations served as a visual indication as to where useful information may be in the documentation — something the original participants in [11] did not have when traversing through Beam’s documentation. For example, P1 in the post task survey stated that Adamite is like “… a mentor who’s [pointing out] where to look for help and some useful pointers from someone who has already been through the process of going through the docs.” Participants still encountered distractor annotations but their shorter length allowed them to be less of a problem than they were in the preliminary study (Section 3.2).

Annotations also helped participants understand which approaches were incorrect, even when ambiguities in the phrasing of the documentation might be misleading. For example, P2 was debating using Partition, a Beam function that partitions a data structure but is inappropriate for completing the task and that control participants were distracted by, but, upon reading an annotation on Partition, she realized this was not the correct path
forward and moved towards a better solution. P7 also reported in his post-task survey that Adamite’s annotations “led [to] me changing my approach, so I’d say they were insightful”.

Participants found Adamite useful for finding additional explanatory information about content in the documentation and for focusing on particular concepts about which they wanted additional information. P9, a successful participant in the task (score = 4), stated:

“The annotations were actually helpful in understanding what to focus on while trying to understand the basics of Apache Beam.”

Participants did not use the advanced filtering mechanisms very often. The only filtering feature they found helpful was filtering by tags, which 2 participants used during the study. P2, in the post task survey, said the ability to filter by tags was a feature she found particularly helpful, while P3 said the filtering by tags was a bit hard since she had difficulty returning to her original view of the annotations (an issue which has been fixed in the current version of Adamite). The visibility around currently selected filters and the relationship between filter and search is something participants got confused about throughout the study, so we added the filter summary at the top of the sidebar as well as a mechanism to revert to default filters.

One feature participants found particularly valuable and that functioned better than Hypothesis was the search feature. Adamite provides live updates character-by-character of the annotations that are displayed from the search query as the user types. In Hypothesis, search only returns annotations once a user clicks “enter” so participants were less sure if they were searching for the right phrase.

Sometimes Adamite did not provide the information participants needed. P6 stated in his post-task survey that Adamite is “good for providing supplemental knowledge, not really for providing the foundational understanding and big picture I feel I needed in this case”. In response, we added support for page level annotations which may be an appropriate place to anchor foundational information such as “getting started” resources.

6.4.3 Survey Results. Participants’ assessment of Adamite’s ease of use and usefulness was mixed to positive (see Table 2, first row). The usefulness of Adamite was rated positive by all but two participants. One participant who thought Adamite would not be useful for their daily work is a user experience researcher who does not program, so they clarified that Adamite does not match the work they normally do. Another participant felt that their programming style would not benefit from Adamite’s annotations, in that they typically utilize code examples and are less interested in reading. However, since much sample code is found using the web, Adamite might already work for annotating code but may benefit from an ability to also support annotations in coding environments.

The users of the tool who expressed mostly positive sentiments called out how it filled in information that was left out of the documentation or expanded upon unclear information, gave them ideas as to where to look next for information, and highlighted the parts of the documentation that were worth reading. These responses nicely match with Agosti’s types of annotations [2] such as interpretation, elucidations, highlight, and navigational aids that Adamite was designed to address.

7 ANNOTATION AUTHORING STUDY WITH ADAMITE

Considering the results of our annotation usage study that high-quality annotations can help participants better utilize an API’s documentation, we wanted to assess whether our additions to Adamite, including authoring support through more advanced annotation types, would help motivated developers create high-quality annotations. Considering the usefulness of Adamite and annotations as a whole requires that the author of the annotation is motivated to create it, we wanted to ensure that other developers, aside from the Adamite creators, could author high-quality annotations.

Between the annotation usage study (Section 6) and this annotation authoring study, some changes were made to Adamite to address previously noted issues, including the filter labels being unclear, an easier way to create
different annotation types, and supporting navigating between the annotation and its position on the page. All figures in the paper and the accompanying video reflect the latest version of Adamite, which is the version that was studied in this annotation authoring study.

7.1 Method

To assess how well developers could author high-quality annotations using Adamite, we completed a user study with 3 parts. Similar to the annotation usage study, we had participants perform training with Adamite by performing an annotation authoring task on the ReactJS documentation which also used to assess the usability and learnability of Adamite. Participants created annotations of each of the annotation types, including a question-type annotation that had an answer on a separate web page that they were supposed to anchor to the original question, pin an annotation, reply to an annotation, create a page-level annotation, add an additional anchor to their first annotation, filter by annotation type, and search for an annotation.

Next, they performed the main task, which was to understand the Beam documentation while completing the coding task described in Section 3.1. We chose to keep the task consistent in order to assess how well the annotations naturally authored by our participants matched the expert annotations from Section 3.2. This also allowed us to derive qualitative differences between how participants used Adamite in comparison to Hypothesis during the first phase of the preliminary study. In this study, similar to the first phase of the preliminary study, we instructed participants to annotate the documentation, ask aloud any questions they had, create question type annotations for their questions, and then to try and find answers to their questions. All annotations that participants created were private so that participants could not see the annotations authored by other participants or the first author for the annotation usage study. Participants were told that the primary focus of the task was not necessarily to complete the coding task but to learn enough using the documentation that they could make reasonable progress. Participants were given one hour for this task. At the end of the study, participants completed the Technology Acceptance Model (TAM) [6] survey and free-response questions about their experience using Adamite. We continued using TAM to see whether the usability and usefulness of our tool differed between using annotated documentation versus authoring annotations.

7.2 Participants

We recruited 8 participants from departmental mailing lists at our university, from online calls for participation on social media sites, and participants with programming backgrounds from the authors’ social circles. Participants are referred to as S1 through S8 to differentiate from prior studies.

All of the participants had some amount of experience using Python, had never used Apache Beam before, and all had been programming for at least 4 years (minimum: 4 years, maximum: 35 years, average: 10.9 years). The participants’ professions included masters students in computer science-related fields, undergraduates in computer science, and professional programmers. The average years of professional software development experience among all participants was 6.1, and 8.1 years among participants who did have professional experience. 1 participant had used distributed data processing APIs before, 1 had never used any data processing or analysis APIs, and all other participants had used other data processing or analysis APIs such as numpy or pandas.

Due to the COVID-19 pandemic, all study sessions were completed remotely using video conferencing software. Participants were audio and video recorded. The study sessions took approximately an hour and a half for each participant. Participants were compensated $15 for their time.

7.3 Analysis

Like the annotation usage study with Adamite, we graded the training task objectively based upon whether participants successfully completed each training task.
Annotations authored during the documentation comprehension task were graded upon how well they matched annotations authored by the first author for the annotation usage task and how well they matched annotations authored by other participants. We graded matching based upon whether the annotation contained subject matter that was similar or the same as the content in our expert annotations. For example, S6 created an annotation on an instance of the term “GroupByKey” with the text “Please provide a code example here,” which matches an annotation made by the first author that provides a code example for using GroupByKey. Matches between participants was graded based upon whether the annotations had the same anchor points and contained similar or the same content. For example, both S2 and S3 made empty annotations on the part of the programming guide that states “[...] a transform does not consume or otherwise alter the input collection”. We used objective criteria to grade how often participants were able to answer their questions, based upon whether the participant marked their question as answered and provided an answer. Additionally, we collected the total number of annotations authored, what types of annotations were authored, and the average length of each authored annotation.

Table 1. Counts and proportions for each annotation type, and overall total.

<table>
<thead>
<tr>
<th>Type</th>
<th>Count</th>
<th>Proportion</th>
</tr>
</thead>
<tbody>
<tr>
<td>Normal</td>
<td>19</td>
<td>15%</td>
</tr>
<tr>
<td>Empty</td>
<td>47</td>
<td>37%</td>
</tr>
<tr>
<td>Issue</td>
<td>17</td>
<td>13%</td>
</tr>
<tr>
<td>Question</td>
<td>37</td>
<td>29%</td>
</tr>
<tr>
<td>To-do</td>
<td>7</td>
<td>6%</td>
</tr>
<tr>
<td>Total</td>
<td>127</td>
<td>100%</td>
</tr>
</tbody>
</table>

7.4 Results and Discussion

7.4.1 Annotation Training. All participants were able to successfully create annotations of each type, reply to an annotation, pin an annotation, mark a question as answered, filter, search, and author a page level annotation. This is an improvement over prior versions of the system that had participants struggling to create annotations of different types and struggling to set their filters.

Some interactions were a bit more complex and were more difficult for participants to complete. Adding a new anchor to an annotation was done successfully by 5/8 participants – the modal interaction of selecting the add-new-anchor button and then selecting in the document as the new anchor point was the reverse of what some participants expected, where they expected to select text first, then add the selected text as a new anchor. Since adding an answer with an anchor utilizes the same unintuitive interaction, participants also struggled with that task (4/8 success rate). We plan to fix this in future Adamite versions.

7.4.2 Documentation Comprehension Task. Participants authored a total of 127 annotations, with each non-empty annotation averaging 8.22 words and each participant, on average, creating 15.9 annotations (see Table 1). Of the 127 annotations, 47 were “empty” annotations, meaning they had no additional content added, leaving 80 annotations with content. Both the average word count and average number of annotations created is an increase over the original preliminary phase 1 study (Section 3.1) with Hypothesis, but this may be contributed to the longer task length for the Adamite authoring study or because the instructions this time more deliberately placed an emphasis on annotation creation. However, given the increased use of the specific annotation types such as issue annotations and question annotations, we also believe that supporting more specific annotation types and adding more structure to the authoring process may have increased the amount of annotations authored between studies (note the small amount of “normal” type annotations in Table 1).

Of the 37 authored question annotations, participants were able to answer 21 of those questions (58%) and 15 of the questions matched questions asked in our Preliminary Phase 2 study of Section 3.2 (41%). The number of answered questions is a large increase in comparison to the percent in the Preliminary Phase 2 study (29%), however it is also a smaller number of asked questions. This may be because they spent more time in the
documentation and thus had less questions about their code, which made up approximately half of the questions from Section 3.2. The Adamite annotators’ questions were also more focused and specific than the typically high-level questions asked in the preliminary study since the current participants were focused on particular parts of the documentation. Moreover, when they were in the code editor, the questions participants asked were not appropriate as annotations (e.g., “What was the goal of this task again?”) which made up 84 of the 138 of the questions asked in Section 3.2. Future versions of Adamite may support question-asking in the code editor to help developers keep track of confusing API aspects and support connecting the relevant parts of the documentation to their questions.

Of the 80 authored annotations that contained some text content, 19 of these matched the content of the expert annotations (26%). Annotations that matched the expert annotations typically referred to important aspects of Beam of which developers should be mindful, and questioned confusing aspects of the documentation such as poorly explained code examples. For instance, a code example showing how to use Beam’s ParDo function shows the return value wrapped in a list which S2 annotated, asking “Why is this returned within a list?” After searching for more information about ParDo, he answered his original question annotation with an anchor on a sentence on a different part of the page that states the function must return an iterable, and added the text “Because [the process function] needs an iterable as an output” – matching the content that the first author annotated. Thus, S2 was able to augment the original documentation with useful, summarized information.

The annotations that did not match the expert-created annotations typically had to do with content that was non-essential to completing the task but was still important and worth investigating. For example, none of the annotations that the first author created could be considered “issue” type annotations that were attempting to call out a problematic aspect of the documentation. Our participants created 12 of these issue annotations, highlighting both incorrect aspects of the documentation (e.g., S7 created an issue annotation that pointed out a part of the documentation that said an example would be provided but was not “Where is this example?”) and areas for improvement in the documentation (e.g., S2 suggested the documentation writers move a description of a method earlier in the documentation because the method is important). Participants also had many questions we had not addressed because prior participants had not asked these questions. These were often questions regarding the high level information about Beam’s pipelines and distributed nature and are fundamental when deciding whether to use Beam and how to successfully use Beam.

Across the 127 annotations authored by our participants, 58 were similar to at least one other annotation (46%). Like the annotations authored by the first author, these annotations typically highlighted important aspects of the documentation, useful resources, and concepts multiple participants struggled with. For example, S2, S4, and S7 all annotated the first instance of the term “FlatMap” with annotations asking what FlatMap is and how it differs from the more common Map function. The first author also made an annotation that discussed the difference between these two transforms. Participants also had annotations that matched one another that the first author had not created annotations for, including highlights reminding themselves and potential future readers that PCollections are immutable, the pipeline structure of Beam may result in a branching pipeline, and highlighting a particularly useful code example showing how to utilize ParDo.

The annotations participants created that did not match other participants annotations typically occurred when participants were in parts of the documentation not many other participants went to or read closely. For example, 9 of S7’s 16 annotations occurred in the Metrics part of the Beam documentation, a part that participants do not need to read in order to succeed in the task. However S7 thought that this section was relevant so she read it closely and annotated it, and we believe that her questions and comments would be useful to other readers of that section. Other participants made annotations that were obviously based on their personal working style, and would not be expected to match other users’ annotations. S5 created annotations on all of the main headers of each section of the page such that they could click on the anchor icon for these annotations to scroll to those parts of the page.
Table 2. Result of the TAM survey on perceived ease of use and usefulness of Adamite between the annotation usage study (Section 6) and the authoring study (Section 7). The answers are: 1 - strongly disagree, 2 - disagree, 3 - more or less disagree, 4 - undecided, 5 - more or less agree, 6 - agree, 7 - strongly agree. Statements on the survey were stated with phrasing like “I would consider Adamite easy to use and easy to achieve what I want.” Significant differences are marked with an asterisk.

<table>
<thead>
<tr>
<th>Easy to Use and Easy to Achieve</th>
<th>Clear How to Use</th>
<th>Easy to Learn</th>
<th>Useful For Daily Work</th>
<th>Enjoyed the Features</th>
<th>Would like for my professional use</th>
<th>Would Recommend</th>
</tr>
</thead>
<tbody>
<tr>
<td>Average (Using annotations)</td>
<td>4.8</td>
<td>5.1</td>
<td>5.7</td>
<td>4.6</td>
<td>5.3</td>
<td>5.1</td>
</tr>
<tr>
<td>Average (Authoring)</td>
<td>5.1</td>
<td>5.5</td>
<td>6.1</td>
<td>5.6</td>
<td>6.1</td>
<td>5.9</td>
</tr>
</tbody>
</table>

Overall, we believe the annotations created by our participants lend support to our hypothesis that highly motivated developers with tool assistance can create high quality annotations that are both useful to themselves and, given the proportion that match our expert annotations and match one another’s annotations, could be beneficial to other developers. Moreover, our developers created annotations that were beneficial that we didn’t previously anticipate, including highlighting important concepts we did not highlight (such as the immutability of Beam’s PCollections) and finding issues within the documentation and suggesting improvements, which could be beneficial for documentation writers and other stakeholders. The small amount of “normal” type annotations suggests that Adamite’s support for multiple types of annotations (especially “question” and “issue” types) can meet developers’ needs when using documentation. Further, the menu that pops up with commonly-asked questions (Figure 2-left) both allowed developers to quickly author annotations that captured their questions, and were used as starting point for participants to structure their questions.

7.4.3 TAM Survey Results. We were interested in seeing how reception of Adamite differed between developers who were primarily using Adamite for authoring, versus the previous group of developers who were using Adamite to utilize annotated documentation. We also wanted to see how our changes to Adamite between these two studies were accepted, so we had our participants in the authoring study complete the same TAM survey as used previously.

Reception to Adamite was overall positive and in comparison to the annotation usage study the scores seemed to have improved and with the difference in recommending Adamite to a colleague reaching statistical significance (unpaired T-test, \( p = .02 \)). Considering participants had to author annotations as opposed to using already-authored annotations that address issues in the documentation, it is somewhat surprising that participants found the tool more useful even though using the tool in this study required more effort. This lends credence to our idea that by providing more annotation authoring support, the act of creating annotations may prove useful for users of Adamite. This is further backed up by our survey response where 6 out of 8 participants reported the question-answer annotation type as their favorite, as they are easy to create with the question-asking menu and allowed developers to cross-link information, with one survey respondent explicitly calling out that they would imagine this linking would help future developers who have the same question.
8 LIMITATIONS

The limitations of all of the user studies reported here are similar to the ones reported in [11]. Given the time constraints and Beam’s complex and difficult subject matter, Adamite may be particularly well-poised to address these shortcomings that might not arise in other, simpler APIs, so the study cannot necessarily be said to apply to simpler situations. Beam’s documentation has also been reported as difficult to use and Beam has a small user-base, so again we have less evidence that Adamite would be useful for APIs with better documentation or APIs with a large user-base that can provide useful crowd-sourced information on Stack Overflow or mailing lists. However, we conjecture that these APIs would still benefit from Adamite in that it may provide short-form, explanatory information that is not typically represented on question-answer sites. Further, our analysis of annotations gathered from Hypothesis (Section 4) suggests that these types of annotations are already being made but could be more popular with the right tooling support. Therefore, an obvious next step with Adamite is to see the role it plays for developers using a more popular API.

Another limitation of our work is that, in the annotation usage studies, developers only used annotations authored by the researcher. While we chose to include annotations that were adopted from Stack Overflow, they still do not fully represent the type of annotations we would expect other users to create. Follow-up work should include using annotations authored by real users.

Lastly, our work is limited by the fact that all of our studies have been lab studies, preventing us from understanding how annotation creation happens organically and whether developers would actually use Adamite on their own in the ways we intended. Our mined Hypothesis annotations provide some evidence that annotation creation does happen in the ways we envisioned, but a large scale field study of Adamite is the only definitive way we can truly know whether developers can use it to make high-quality annotations without a researcher present. Adamite uses the Google Cloud APIs as the backend database, so it could theoretically scale to an arbitrary number of users, so we would like to do a large field-trial of using Adamite by real programmers on real tasks.

9 CONCLUSION

Our preliminary studies, development of Adamite and its user study together provide evidence that annotations may be beneficial for helping to mitigate some of the well-known shortcomings of API documentation. In particular, Adamite helped developers to produce more accurate and correct code, to better utilize documentation for learning difficult concepts, and to find the most useful and relevant parts of the documentation for their learning needs. When using Adamite to author annotations, developers were able to keep track of their open questions and close the questions out with useful answers, keep track of their to-do items, and point out problematic aspects of the documentation with suggestions for improvement. While there is still room for improvement, Adamite lends support to the idea that developers may effectively utilize other developers’ short-form information and augment the underlying documentation with useful information.

REFERENCES


