Support for Long-Form Documentation Authoring and Maintenance

Amber Horvath
Human-Computer Interaction Institute
Carnegie Mellon University
Pittsburgh, PA, USA ahorvath@cs.cmu.edu

Andrew Macvean
Google
Seattle, WA, USA
amacvean@google.com

Brad A. Myers
Human-Computer Interaction Institute
Carnegie Mellon University
Pittsburgh, PA, USA bam@cs.cmu.edu

***Abstract*—When creating a software project of any significant size or complexity, developers must write about the code in some form. While this is a common practice, tooling support for creating, and perhaps more significantly, maintaining these documents remains limited, despite these documents benefiting later users of the corresponding code. In order to combat some of these known challenges, we developed Sodalite – a context-aware, Visual Studio Code extension for writing and maintaining code-related documents. Sodalite represents a holistic approach to the documentation authoring, maintaining, and using life cycle by suggesting relevant code to attach to the document using common code-writing templates, evaluating how up-to-date the document is using the code in the editor, automatically attempting to identify out-of-date information, and, based on how successful the system is, notifying the user of how trustworthy the document seems to be given their version of the related code. In our preliminary evaluation of Sodalite, we found that users of the system were able to successfully author documents about their code and the system's out-of-date link checking was accurate 86.5% of the time.**

***Index Terms*—documentation, software maintenance**

I. Introduction

Modern software engineering often requires developers to write about their code, including design documents, onboarding instructions for new developers, change logs, and pull requests and issue descriptions on GitHub. It is uncommon to create a software project of any significant size or complexity without writing about it in some form, given that these documents can be helpful for others [1]–[4].

Despite the benefits of these long-form documents, hereafter referred to as "stories", it is uniquely difficult to document [5], [6] and communicate about code [7]. Some of the challenges when *authoring* these documents include establishing a shared context about which parts of the code are related to the writing [7], ensuring that the code and the writing about the code are kept in sync with one another [8], and knowing what does and does not need to be documented [4], [5], [8]. Once a code story has been written, challenges still remain for the author with documents written about the code going out-of-date [4], [8], [9], with code changes causing 39% of documentation content issues [8].

Readers of code stories face their own challenges. Beyond out-of-date information [8], [10], other issues include missing information [4], [5], [8], [11], [12], and a lack of context [13]. These compounding issues can lead to developers getting blocked [4], [5], [14], [15], and to introducing avoidable bugs [16], [17].

In this work, we attempt to address some of these challenges around authoring, maintaining, and using stories about code with our system, Sodalite[1]. Sodalite is a Visual Studio Code extension [18] which currently supports the JavaScript and TypeScript [19] programming languages. To assist the developer in *authoring* their code story, Sodalite leverages the context of the IDE to help structure documentation and discover and create links between the text in their story and the relevant code in the IDE. The system then helps the author *maintain* their code stories by automatically detecting code links that are no longer valid and, if appropriate, attempts to reconnect the link. For *readers* of the story, Sodalite uses similar mechanisms to ensure that the links are valid and, if not, warn the user and show what the code looked like at the time in which the story was authored. Through this approach, we hypothesize that Sodalite can help developers author and maintain useful documents for themselves and other developers working on a repository together.

Our work makes the following contributions:

- Identification of the kinds of code links that can be used to enrich code stories.
- Our system, Sodalite, that helps developers author, maintain and use long-form writings about code by:
  - Supporting authors in identifying relevant parts of code through recommending related code.
  - Assisting maintainers through using the context of the editor to identify references that are out-of-date, which is a significant reason code documentation becomes problematic [8].
  - Helping readers find important documents and to identify what parts of the code lack trustworthiness, along with providing them enough context about the originally authored story to help them ascertain the original intent of the writing.

[1]Sodalite is a mineral, and here stands for **S**tories for **O**n-boarding as **D**ocumentation **A**uthoring, **L**everaging **I**DEs for **T**ext **E**nhancements.

II. Related Work

A. Writing About Code

Prior work about writing software documentation have ranged from understanding what information is in documentation [20], [21], what the problems are with that information [5], [8], [11], [14], [15], [22]–[27], how software documentation is authored [3], [12], [28], and automating documentation processes to offset authoring costs and standardize the information present in documentation [29], [30]. Notably, the majority of this documentation work is in the context of software documentation for end users of a software library, e.g., API documentation [31]. Most relevant to our work are studies about documentation created for other developers working on the *same code base*, such as internal documentation about the code base [4], [12], where the primary goal is to help other developers understand and contribute to the code base. Our work expands upon this work by contributing a system designed specifically to help create this type of documentation, with a focus on combating some of the known issues in authoring and maintaining this documentation.

B. Tools for Writing About Code

To assist with some of the aforementioned challenges of writing about code, some research projects have attempted to make *authoring* information about code easier. Some projects have focused on developing alternative, spatially organized development environments in which the documentation and source code can be kept together [32]–[34]. Others have focused on specific types of documentation such as code tours [35], notes [36], or tutorial generation [37]. Our work differs by adopting an already-popular programming paradigm and supporting the maintenance of the documentation.

C. Maintaining and Using Documentation

Once some software documentation is made, researchers have studied how developers maintain those documents and use that information. In studies of usage, researchers have identified many problems of documentation that lead to the documentation being less trustworthy [38], including questions about how up-to-date the information is [8], [14], [22] and how complete [5], [15]. A survey of developers at one company reported that they rarely updated documentation and that they correctly assumed that documentation content is out-of-date [22]. One reason for this lack of maintenance is that finding the appropriate places to update given a change can be challenging, with developers reporting that they would value a tool that helps identify those locations [39]. Our system attempts to reduce some of these costs by having maintenance be a core design consideration by automatically locating and highlighting the out-of-date portions of the document given which code was changed.

III. Overview and Design of Sodalite

In order to use Sodalite, a user begins by opening the Sodalite window in Visual Studio Code which functions like any other pane in the editor (i.e., it can be dragged, resized, closed, etc.). The user can then view the stories that have been associated with the user's currently-open GitHub repository or choose to author a new story (see Figure 1a). When authoring a new story, the user will be presented with a rich text editor, hereafter referred to as the "story editor" (in contrast to the "code editor", which refers to the Visual Studio Code IDE), and users can utilize *code story templates* (a set of common documentation types developers can use to structure their documentation) and *code links*.

A. Code Links and Suggestions

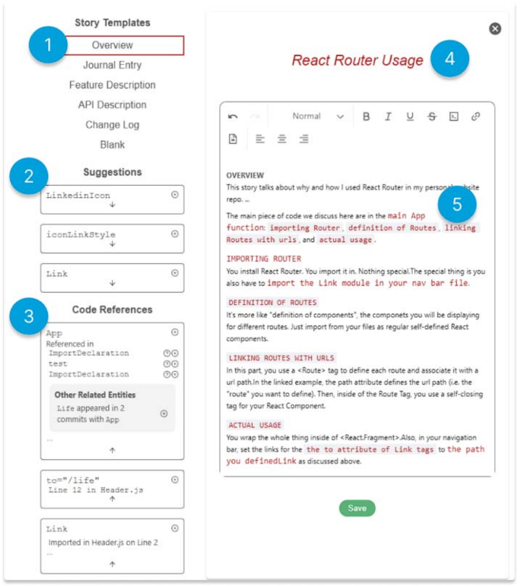
A key feature of Sodalite are *code links*, where the story references the code. Links can be made using the "Suggestions" pane (see Figure 1a-2), which lists code links that the author may include in their code story. Code links come in three varieties:

- *Identifier definitions*, which link to where a specific code entity is first defined. A user can create an identifier definition link by selecting the "plus" button in the suggestions pane at the top right corner of the code link box. Identifier definitions also include additional information about the identifier, including places in which it is referenced, and other classes or functions it references. The first code link in Figure 1a-3 is an identifier definition.
- *Identifier references*, which link to a specific instance in which a particular identifier is used. An author can make an identifier reference by selecting a reference that is listed in an identifier definition's list of referenced locations. The third code link in Figure 1a-3 is an identifier reference.
- *Code ranges*, which can be any arbitrary range of code that the user has selected in the code editor. The second code link in Figure 1a-3 is a code range.

We designed the code links to accommodate different levels of granularity when discussing code. Sodalite supports code ranges, since prior research has indicated that developers often refer to code at the snippet level [40]. We support identifier definitions and references, as that is the granularity often used in API documentation.

Code links can be represented in a code story in two different ways. If the user has no text selected in the story editor, the code link will either insert the name of the identifier or the code range's content at the location of the user's cursor. If the user has selected some text in their code story, that text will be linked to their code (see Figure 1a-5). Either way, clicking on a code link within the story editor will navigate to wherever that particular code link is located in the code editor. Once a code link has been added to a story, it will appear in a "Code References" list (Figure 1a-3), such that the link may be used elsewhere in the story.

Code links can also contain additional meta-data Sodalite was able to determine about that part of the code. This includes, for the identifiers, where they are defined and referenced in different parts of the code. Once a code link has been included in the code story the "Suggestions" pane will include



React Router Usage

OVERVIEW
This story talks about why and how I used React Router in my personal website repo. ...

The main piece of code we discuss here are in the `main App function`: `importing Router`, `definition of Routes`, `linking Routes with urls`, and `actual usage`.

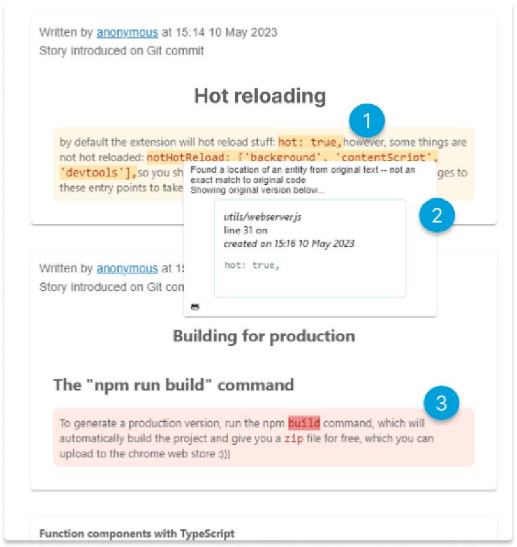
IMPORTING ROUTER
You install React Router. You import it in. Nothing special.The special thing is you also have to `import the Link module in your nav bar file`.

DEFINITION OF ROUTES
It's more like "definition of components", the componets you will be displaying for different routes. Just import from your files as regular self-defined React components.

LINKING ROUTES WITH URLS
In this part, you use a <Route> tag to define each route and associate it with a url path.In the linked example, the path attribute defines the url path (i.e. the "route" you want to define). Then, inside of the Route Tag, you use a self-closing tag for your React Component.

ACTUAL USAGE
You wrap the whole thing inside of <React.Fragment>.Also, in your navigation bar, set the links for the `the to attribute of Link tags` to `the path you definedLink` as discussed above.

(a) Authoring with Sodalite



Written by anonymous at 15:14 10 May 2023
Story introduced on Git commit

Hot reloading

by default the extension will hot reload stuff: `hot: true,` however, some things are not hot reloaded: `notHotReload: ['background', 'contentScript', 'devtools'],` so you sh ... ges to these entry points to take

Found a location of an entity from original text -- not an exact match to original code
Showing original version below...

utils/webserver.js
line 31 on
created on 15:16 10 May 2023

```
hot: true,
```

Written by anonymous at 1
Story introduced on Git con

Building for production

The "npm run build" command

To generate a production version, run the npm `build` command, which will automatically build the project and give you a `zip` file for free, which you can upload to the chrome web store :)))

Function components with TypeScript

(b) Reading and Maintaining with Sodalite

Fig. 1: How Sodalite appears when authoring a story and post-authoring. For Authoring (a), (1) a story template a user can choose, with “Overview” selected, (2) the current code link suggestions, (3) the already-included code links, (4) the title. (5) the rich text editor, where the red text represents code links. For Reading and Maintaining (b), (1) a code story, with the yellow showing that the content is in need of review, (2) the popup that appears when hovering over a code link that is in need of review. (3) a code story with invalid code links (shown red).

other identifiers that were commonly edited at the same time as that particular identifier. We identify these “co-edits” by parsing the Git commit history for that particular identifier and count when other identifiers appear in the same commit. In this way, we attempt to identify parts of code that are related but not in a way AST parsing would find.

Given these different types of code links, the “Suggestions” pane uses different sources of information to populate its list. Sodalite examines both what the user is doing in the story editor and what they are doing in the code editor. The information that Sodalite leverages from the story includes what references are already included in the story and what the user has most recently typed. Identifiers related to references already in the code story will be prioritized, as will identifiers that match some part of the most recently-typed text. The system then complements that information with what it knows about the user’s current location within the code, including if the user is currently selecting an identifier, and, if so, what identifiers are related to the selected identifier.

When a story is saved, Sodalite generates a JSON file in a system-generated folder, which can be committed to the user’s Git repository and used by other programmers. This JSON file is also used by the system when determining whether code links are out-of-date.

B. Support for Reading

Sodalite has some features designed specifically to help readers of code stories better understand and utilize the documentation. A core feature of Sodalite for readers of code stories is the fact that it is situated within the context of the code. Developers have previously stated they value source code and code comments more than other types of documentation [41] – we hypothesize that bringing the type of information typically in external documentation into the code editor will allow for “the best of both worlds” by staying within the developer’s working context while also supporting longer-form documentation.

Another feature of Sodalite that we expect to help users of code stories are the bi-directional nature of the code links. When some code has been linked in a story, the code will be highlighted within the IDE, such that users of the code can discover pertinent documentation that is relevant to that code. The highlighted code brings up a hover text that shows a preview of the code story, including the surrounding text from the story, the part of the story that is linked to the code, and the name and author of the story. If the surrounding story text contains code links to other parts of the code, those links will navigate the user to the code link’s location. If the user clicks on the story link in the hover text, the Sodalite pane will open and scroll to the correct part of the story.

C. Maintenance Support

Sodalite leverages being in the code editor such that it can mark parts of stories as “valid” (in which all code links are valid), potentially in “need of review” (in which the system found a potential match for the code link – see Figure 1b-2) or definitely "invalid" based on how well the system is able to match the code references in the story to the code currently in the user's project in Visual Studio Code[2].

On launch, Sodalite parses every code story file in the user's current project and builds an internal AST representation of the code in the project to compare the code links against. We use the different types of code links to inform how to re-attach a particular link and whether that attachment is problematic.

For code anchors, given their varying content (e.g., anything from a string to a full multi-line expression), we begin by evaluating whether the position we have saved contains the same code as the code link. If not, we then look for whether the code from the code link exists anywhere within the document. If that does not work, we use purely text-based matching mechanisms to discern candidate matching points, since that was the most successful method used in [42]. We weigh the probability that the location is correct using a combination of calculating the edit-distance between the two versions and the surrounding code, and the difference between the candidate line(s) of code and the original location. If there is a low score or no matches, we mark the anchor as invalid and the corresponding text in the story as in need of review. To assist the author in finding either a new location to link the story text to in the code or to discard the link, we present additional metadata about the code (see Figure 1b-2 popover).

For definition references, the system searches the internal representation to see whether the identifier is defined at its last known location. If the definition is not there, the search will expand outwards to see if the identifier is defined in a different file and, if so, attaches to that location, but marks the link as potentially invalid, considering it may be a different identifier that just happens to have the same name. If the system cannot find a definition for the particular referenced code entity, the system uses the text-based re-anchoring algorithm. If the algorithm does not return a result of a sufficiently highly-weighted likelihood, it marks the reference as invalid, and the surrounding text within the code story as in need of review by the author.

A similar approach is used for checking identifier references in the code. We begin by seeing whether there are one or more references in the code in the last-known scope in which the reference was used. If there are multiple candidate matches, we find the reference with the most similar AST path to the saved code anchor information. If there are no candidate matches, we once again fall back on the text-matching algorithm and, if the result is inadequate, mark the text within the story as potentially invalid and in need of review.

IV. Preliminary Evaluation of Sodalite

In order to assess the design concepts used in Sodalite, we ran a preliminary user study. Participants were instructed to select a JavaScript or TypeScript project of their choosing and author a code story using Sodalite. Upon completion of the authoring session, the first author and the participant reverted their code to an earlier version[3] to see how well the maintenance system worked in identifying valid versus invalid code links. We ran 4 participants, each session took about 90 minutes, participants were compensated \$25 for their time, and the study was approved by our institution's Institutional Review Board.

In our preliminary evaluation, we found that participants were able to successfully use Sodalite and its features to document their code. Sodalite also succeeded in determining valid, invalid, and in need of review code links 86.5% of the time. The cases in which Sodalite failed were due to the system either missing a link due to the text and location being too dissimilar or due to the original code being too generic to find one singular attachment point – both of these cases could be combated by adding in more heuristics about the code and its surrounding context for the re-anchoring algorithm to utilize. Nonetheless, these results lend preliminary support to our design choices in Sodalite.

V. Discussion and Future Work

An obvious next step is to run a more comprehensive user study to see how Sodalite authoring and usage generalizes across more developers and their code. Our preliminary study also did not evaluate if later readers found the code stories useful or not. A potential study design may be to have one set of developers author a code story on one version of a code base, then have another set of developers on a different version of the same code base attempt to use the authored code stories to complete a programming task, such that every aspect of Sodalite is evaluated.

A current limitation of Sodalite is that all of the mechanisms for determining whether or not the story is out-of-date are contingent upon the story including code links. The expectation is that, given Sodalite being located within the IDE, developers will naturally utilize that context and reference their code within their code stories. Future versions of Sodalite may benefit from additional mechanisms for assessing the validity of the text in comparison to the code, for example, through leveraging crowd-sourcing mechanisms for quality control [3], [43].

We see Sodalite as an example of how leveraging a developer's working context can be used to address long-standing issues in software engineering, specifically documentation. By understanding what the code the user is working on, Sodalite was able to mark documentation as valid or invalid 86.5% of the time, which would not be possible for static documentation hosted on e.g., a web page. With the recent rise of large language models (LLMs), this personalized and contextualized approach to documentation could be further improved through creating and summarizing documentation using LLMs that are aware of the user's working context.

[2]There are situations in which the documentation may go out-of-date that our system would not capture, but, a study of documentation problems [8] found that most cases in which the documentation went out-of-date was due to the code changing, so we focus on that case with Sodalite.

[3]We considered requiring participants to document an old version of their code but developers do not typically document old code and they may not remember their old code well enough to document it.

Acknowledgments

This research was funded in part by the NSF under grant CCF-2007482 and by gifts from Google. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the sponsors. In addition, we would like to thank our participants and our reviewers for their thoughtful input and feedback.

References

[1] A. Ju, H. Sajnani, S. Kelly, and K. Herzig, "A case study of onboarding in software teams: Tasks and strategies," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pp. 613–623, IEEE, 2021.

[2] A. Begel and B. Simon, "Novice software developers, all over again," in *Proceedings of the Fourth International Workshop on Computing Education Research*, ICER '08, (New York, NY, USA), p. 3–14, Association for Computing Machinery, 2008.

[3] B. Dagenais and M. P. Robillard, "Creating and evolving developer documentation: Understanding the decisions of open source contributors," in *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE '10, (New York, NY, USA), p. 127–136, Association for Computing Machinery, 2010.

[4] M. P. Robillard, "Turnover-induced knowledge loss in practice," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2021, (New York, NY, USA), p. 1292–1302, Association for Computing Machinery, 2021.

[5] M. P. Robillard and R. DeLine, "A field study of API learning obstacles," *Empirical Software Engineering*, vol. 16, pp. 703–732, 2011.

[6] C.-A. Postava-Davignon, C. Kamachi, C. Clarke, G. Kushmerek, M. B. Rettger, P. Monchamp, and R. Ellis, "Incorporating usability testing into the documentation process," *Technical Communication*, vol. 51, pp. 36–44, Feb. 2004.

[7] S. Oney, C. Brooks, and P. Resnick, "Creating guided code explanations with chat.codes," *Proc. ACM Hum.-Comput. Interact.*, vol. 2, nov 2018.

[8] E. Aghajani, C. Nagy, O. L. Vega-Márquez, M. Linares-Vásquez, L. Moreno, G. Bavota, and M. Lanza, "Software documentation issues unveiled," in *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, (Montreal, QC, Canada), pp. 1199–1210, IEEE, 2019.

[9] M. P. Robillard, "What makes apis hard to learn? answers from developers," *IEEE Software*, vol. 26, pp. 27–34, Oct. 2009.

[10] H. Zhang, S. Wang, T.-H. Chen, Y. Zou, and A. E. Hassan, "An empirical study of obsolete answers on stack overflow," *IEEE Transactions on Software Engineering*, vol. 47, no. 4, pp. 850–862, 2021.

[11] E. Aghajani, C. Nagy, M. Linares-Vásquez, L. Moreno, G. Bavota, M. Lanza, and D. C. Shepherd, "Software documentation: The practitioners' perspective," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ICSE '20, (New York, NY, USA), p. 590–601, Association for Computing Machinery, 2020.

[12] Y. Shmerlin, I. Hadar, D. Kliger, and H. Makabee, "To document or not to document? an exploratory study on developers' motivation to document code," in *Advanced Information Systems Engineering Workshops: CAiSE 2015 International Workshops, Stockholm, Sweden, June 8-9, 2015, Proceedings 27*, pp. 100–106, Springer, 2015.

[13] J.-C. Chen and S.-J. Huang, "An empirical analysis of the impact of software development problem factors on software maintainability," *Journal of Systems and Software*, vol. 82, no. 6, pp. 981–992, 2009.

[14] G. Uddin and M. P. Robillard, "How API documentation fails," *IEEE Software*, vol. 32, pp. 68–75, Aug. 2015.

[15] T. Roehm, R. Tiarks, R. Koschke, and W. Maalej, "How do professional developers comprehend software?," in *ICSE 2012*, (New York, NY, USA), pp. 632–542, ACM, 2012.

[16] S. Fahl, M. Harbach, H. Perl, M. Koetter, and M. Smith, "Rethinking ssl development in an appified world," in *Proceedings of the 2013 AC SIGSAC Conference on Computer and Communications Security*, CCS '13, (New York City, NY, USA), pp. 49–60, ACM, 2013.

[17] T. Zhang, G. Upadhyaya, A. Reinhardt, H. Rajan, and M. Kim, "Are code examples on an online q&a forum reliable? a study of api misuse on stack overflow," in *ICSE 2018*, (New York, NY, USA), pp. 886–896, ACM, 2018.

[18] Microsoft, "Visual studio code," 2022.

[19] Microsoft, "Typescript: Javascript with syntax for types.," 2023.

[20] W. Maalej and M. P. Robillard, "Patterns of knowledge in api reference documentation," *IEEE Transactions on Software Engineering*, vol. 39, no. 9, pp. 1264–1282, 2013.

[21] A. Head, C. Sadowski, E. Murphy-Hill, and A. Knight, "When not to comment: Questions and tradeoffs with api documentation for c++ projects," in *Proceedings of the 40th International Conference on Software Engineering*, ICSE '18, (New York, NY, USA), p. 643–653, Association for Computing Machinery, 2018.

[22] T. C. Lethbirdge, J. Singer, and A. Forward, "How software engineers use documentation: the state of the practice," *IEEE Software*, vol. 20, pp. 35–39, Nov. 2003.

[23] J. Nykaza, R. Messinger, F. Boehme, C. L. Norman, M. Mace, and M. Gordon, "What programmers really want: results of a needs assessment for sdk documentation," in *SIGDOC 2002*, (New York, NY, USA), pp. 133–141, ACM, 2002.

[24] M. Meng, S. M. Steinhard, and A. Schubert, "How developers use api documentation: an observation study," *Communication Design Quarterly*, vol. 7, pp. 40–49, 2019.

[25] S. Endrikat, S. Hanenberg, R. Robbes, and A. Stefik, "How do api documentation and static typing affect api usability?," in *ICSE 2014*, (New York City, NY, USA), pp. 632–642, ACM, 2014.

[26] P. Chatterjee, M. A. Nishi, K. Damevski, V. Augustine, L. Pollock, and N. A. Kraft, "What information about code snippets is available in different software-related documents? an exploratory study," in *SANER 2017*, (New York City, NY, USA), pp. 382–386, IEEE, 2017.

[27] A. Horvath, M. X. Liu, R. Hendriksen, C. Shannon, E. Paterson, K. Jawad, A. Macvean, and B. A. Myers, "Understanding how programmers can use annotations on documentation," in *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, CHI '22, (New York, NY, USA), Association for Computing Machinery, 2022.

[28] C. J. Stettina and W. Heijstek, "Necessary and neglected? an empirical study of internal documentation in agile software development teams," in *Proceedings of the 29th ACM International Conference on Design of Communication*, SIGDOC '11, (New York, NY, USA), p. 159–166, Association for Computing Machinery, 2011.

[29] S. Haiduc, J. Aponte, L. Moreno, and A. Marcus, "On the use of automated text summarization techniques for summarizing source code," in *2010 17th Working Conference on Reverse Engineering*, pp. 35–44, IEEE, 2010.

[30] E. Aghajani, G. Bavota, M. Linares-Vásquez, and M. Lanza, "Automated documentation of android apps," *IEEE Transactions on Software Engineering*, vol. 47, no. 1, pp. 204–220, 2019.

[31] B. A. Myers and J. Stylos, "Improving api usability," *Communications of the ACM*, vol. 59, no. 6, pp. 62–69, 2016.

[32] M. Adeli, N. Nelson, S. Chattopadhyay, H. Coffey, A. Henley, and A. Sarma, "Supporting code comprehension via annotations: Right information at the right time and place," in *2020 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, pp. 1–10, IEEE, 2020.

[33] J. Brandt, P. J. Guo, J. Lewenstein, M. Dontcheva, and S. R. Klemmer, "Two studies of opportunistic programming: Interleaving web foraging, learning, and writing code," in *CHI '09*, CHI '09, (New York, NY, USA), p. 1589–1598, Association for Computing Machinery, 2009.

[34] A. Bragdon, R. Zeleznik, S. P. Reiss, S. Karumuri, W. Cheung, J. Kaplan, C. Coleman, F. Adeputra, and J. J. LaViola, "Code bubbles: A working set-based interface for code understanding and maintenance," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, (New York, NY, USA), p. 2503–2512, Association for Computing Machinery, 2010.

[35] G. Taylor and S. Clarke, "A tour through code: Helping developers become familiar with unfamiliar code," in *Psychology of Programming Interest Group 33rd Annual Workshop*, PPIG 2022, pp. 114–126.

[36] A. Horvath, B. Myers, A. Macvean, and I. Rahman, "Using annotations for sensemaking about code," in *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*, UIST '22, (New York, NY, USA), Association for Computing Machinery, 2022.

[37] A. Y. Wang, A. Head, A. Zhang, S. Oney, and C. Brooks, "Colaroid: A literate programming approach for authoring explorable multi-stage tutorials," in *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI '23.

[38] M. X. Liu, A. Kittur, and B. A. Myers, "To reuse or not to reuse? a framework and system for evaluating summarized knowledge," *Proc. ACM Hum.-Comput. Interact.*, vol. 5, apr 2021.

[39] A. Forward and T. C. Lethbridge, "The relevance of software documentation, tools and technologies: A survey," in *Proceedings of the 2002 ACM Symposium on Document Engineering*, DocEng '02, (New York, NY, USA), p. 26–33, Association for Computing Machinery, 2002.

[40] R. Holmes and A. Begel, "Deep intellisense: a tool for rehydrating evaporated information," in *Proceedings of the 2008 international working conference on Mining software repositories*, pp. 23–26, 2008.

[41] S. C. B. de Souza, N. Anquetil, and K. M. de Oliveira, "A study of the documentation essential to software maintenance," in *Proceedings of the 23rd Annual International Conference on Design of Communication: Documenting amp; Designing for Pervasive Information*, SIGDOC '05, (New York, NY, USA), p. 68–75, Association for Computing Machinery, 2005.

[42] S. P. Reiss, "Tracking source locations," in *Proceedings of the 30th International Conference on Software Engineering*, ICSE '08, (New York, NY, USA), p. 11–20, Association for Computing Machinery, 2008.

[43] M. Allahbakhsh, B. Benatallah, A. Ignjatovic, H. R. Motahari-Nezhad, E. Bertino, and S. Dustdar, "Quality control in crowdsourcing systems: Issues and directions," *IEEE Internet Computing*, vol. 17, pp. 76–81, 2013.