Unakite: Scaffolding Developers' Decision-Making Using the Web

Michael Xieyang Liu[1], Jane Hsieh[2], Nathan Hahn[1], Angelina Zhou[1], Emily Deng[1], Shaun Burley[1], Cynthia Taylor[2], Aniket Kittur[1], Brad A. Myers[1]
[1]Human-Computer Interaction Institute, Carnegie Mellon University, Pittsburgh, PA
[2]Department of Computer Science, Oberlin College, Oberlin, Ohio
{xieyangl, nhahn, nkittur, bam}@cs.cmu.edu, {ajzhou, edeng}@andrew.cmu.edu,
{jhsieh, ctaylor}@oberlin.edu, me@shaunburley.com

ABSTRACT
Developers spend a significant portion of their time searching for solutions and methods online. While numerous tools have been developed to support this exploratory process, in many cases the answers to developers' questions involve trade-offs among multiple valid options and not just a single solution. Through interviews, we discovered that developers express a desire for help with decision-making and understanding trade-offs. Through an analysis of Stack Overflow posts, we observed that many answers describe such trade-offs. These findings suggest that tools designed to help a developer capture information and make decisions about trade-offs can provide crucial benefits for both the developers and others who want to understand their design rationale. In this work, we probe this hypothesis with a prototype system named Unakite that collects, organizes, and keeps track of information about trade-offs and builds a comparison table, which can be saved as a design rationale for later use. Our evaluation results show that Unakite reduces the cost of capturing tradeoff-related information by 45%, and that the resulting comparison table speeds up a subsequent developer's ability to understand the trade-offs by about a factor of three.

Author Keywords
Programming Support Tools; Trade-offs; Decision making

CCS Concepts
•**Information systems** → **Decision support systems;** •**Software and its engineering** → **Software design trade-offs;**

INTRODUCTION
Developers spend a significant portion of their time searching the web for answers [16, 60]. Past HCI and software engineering research supporting developers' foraging has focused on helping developers find a specific solution such as example code [56, 15, 55] and API documentation [64, 60], integrating it into one's own code [50, 70], and linking it back to the source [15, 50]. However, for many programming problems, there is no single correct solution – instead, there are many valid possible options (each with different trade-offs), and the decision comes down to how well each option matches the developer's goals [52, 28, 57, 54, 42, 59, 43]. For decision problems such as picking a JavaScript library to build websites, choosing an encryption algorithm to hash users' passwords, or seemingly straightforward ones like how to draw a blue circle on a web page, there is more than one good answer and trade-offs exist among all of the valid alternatives. For example, when picking a deep learning framework, Tensorflow [11] (with its top-notch performance and scalability) may be more suitable for building large commercial AI systems, while a more approachable framework like PyTorch [10] may be a better choice for small academic projects and experiments.

As the number of frameworks, libraries, languages, and patterns increases [7, 8, 9], evidence about the trade-offs often needs to be collected across many competing information sources (e.g., documentation sites, blog posts, and discussion threads), and synthesized so that the developer can make an informed decision. Currently, this is a challenging process since there are high costs involved in capturing content, maintaining its provenance (its source), and synthesizing it with other content (that may very well be in different formats and structures) in a way that helps the developer to make a decision. For example, one developer in our formative study reported exactly these problems when copy-and-pasting relevant information into a Google Doc while deciding between using React [24] or Angular [30] to build her personal website.

This issue is compounded when later developers try to use the initial developers' code and discover that they need to understand why and how the original decision was made. Without proper documentation, it is hard for subsequent readers to figure out the context of the decision space: what options were considered, what criteria or constraints should be met, what the resulting trade-offs are, and what was deemed to be the most important and why. Indeed, understanding such *design rationale* is cited as one of the hardest questions for developers to answer about unfamiliar code [40, 63, 41].

UIST '19, October 20-23, 2019, New Orleans, LA, USA.

ACM ISBN 978-1-4503-6816-2/19/10 ...$15.00.
http://dx.doi.org/10.1145/3332165.3347908

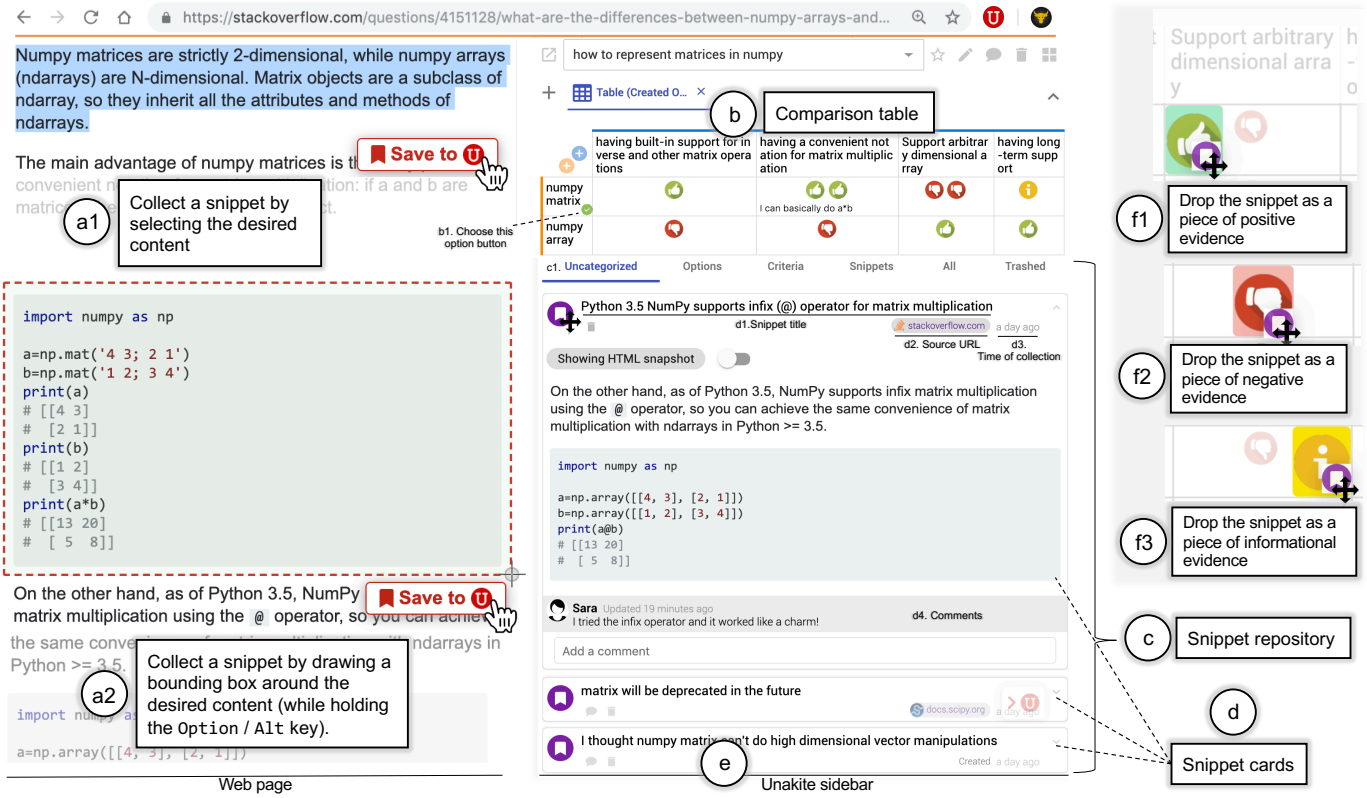

Figure 1: Unakite's user interfaces. With Unakite, a developer collects a snippet by selecting the desired content (a1) or by drawing a bounding box around the desired content (while holding the `Option` / `Alt` key) (a2) and clicking the "Save to U" button. The collected snippet immediately shows up under the "Uncategorized" tab in the snippet repository (c) as a snippet card (d) inside the Unakite sidebar (e), which shows the current task at the top ("how to represent matrices in numpy") along with the drop-down menu to pick other tasks and various tools for the task. The developer can quickly drag the snippet and drop it in one of the cells in the comparison table near the top (b). (f1-f3) show the details of the three parts of each cell in the table where the snippet can be dropped.

In needs-finding interviews with 15 developers, we found that they expressed a desire for help with decision making and understanding others' design rationale when presented with decision problems involving multiple trade-offs. Next, we analyzed Stack Overflow (SO) questions, which revealed that many answers on SO contain information describing such trade-offs. These findings indicate that there are potential benefits to tools that help developers capture information, make decisions, and save the context for future reference.

To investigate the validity of this hypothesis, we built a prototype system called Unakite[1] as a plugin for the Chrome browser. Unakite reduces the costs of capturing and organizing information about trade-offs, and persists this information so that it can serve as the design rationale. To reduce the burden on developers, Unakite provides these capabilities *while the user is searching and browsing*. Unakite is named after a pink and green semi-precious stone, and stands for "Users Need Accelerators for Knowledge for Implementations in Technology Environments". It enables developers to easily collect content from any web page into an information repository. The amassed information is organized in a tabular format (which we selected based on evidence from our formative studies) that crystallizes the trade-offs among various solutions *in situ*. The resulting organizational structures are automatically preserved and can be shared to support collaboration, documentation, and integration with code through comments.

We evaluated how well Unakite can support participants in collecting and organizing information about trade-offs as well as in understanding such gathered content. Compared to using Google Docs to build and maintain a comparison table, Unakite reduces the overhead cost of capturing tradeoff-related information by 45%. Compared to just going through unstructured information on a set of web pages, participants using Unakite were able to understand trade-offs involved in previously-made decisions about three times faster.

The primary contributions described in this paper include:

- formative studies showing developers' needs for support with decision-making,
- Unakite, a novel system that reduces the costs of capturing and organizing online information and preserves the knowledge as design rationale, and
- an evaluation of Unakite through two controlled studies that offer insights into its usability, usefulness, and effectiveness.

RELATED WORK

Programming Support Tools for Finding Information

Many previous systems attempt to help developers find a specific piece of information. For example, tools like Mica [64],

[1] Unakite is available at **`https://unakite.info`**

Assieme [33], and Libra [56] improve existing general-purpose search engines by concentrating programming-specific information to help developers locate the most relevant API choices or other resources; BluePrint [15] and Seahawk [55] mined software repositories and online Q&A forums to form example-centric code searches; and CodeOn [19, 20] explores the possibility of offloading the search and problem-solving job to remote helpers. Other systems keep track of the sources that developers use to support going back to them [64, 15]. However, theories and empirical work have long pointed out that finding relevant information is just the first step [58, 69] in such complex sensemaking tasks. Unlike previously mentioned systems, Unakite focuses on helping developers collect and synthesize relevant information into structured knowledge, which is arguably the next important step towards an actual understanding of the decision space [31].

Design Rationale In Software Engineering

Prior research has identified that understanding the intent and rationale for why code was done in a particular way is one of the hard-to-answer questions for developers [40, 63, 41]. LaToza et al. [39] suggested that developers often try to understand the reasons behind surprising decisions by deducing the possible motivating requirements and criteria. Ko et al. [38] reported that developers frequently speculate about the correctness and legitimacy of a decision, and that they often wish to see the alternatives and their trade-offs that were considered.

Despite the prevalence of the problem, effective support for understanding decisions in programming is still considered an open question. LaToza et al. [40] suggested that while some of the questions could be tackled by changing and testing the code itself, the majority of design decision questions are difficult to answer in this fashion due to their non-functional nature. Asking colleagues and teammates might help ease the underlying concern about design decisions, alternatives, and criteria that are nearly impossible to test [41], but often the original designers are not available. Unakite addresses this problem by keeping track of the initial developers' decision making trails in a structured way so that it provides a springboard and a consistent narrative for later code readers to easily resurrect and assimilate the author's original design rationale.

Code Comments & Documentation

Comments serve to improve source code readability, and are considered valuable for code understanding and maintenance [25, 65], which suggests that they are good locations for documenting the rationale behind code decisions [40]. However, it is well-known that developers do not like to write comments (even if they are simple) [66, 51], and do not trust comments to be up-to-date [26]. Another recent idea is to permit developers to annotate software projects with rich media like pictures and audio [45] as the design rationale, which may reduce the cost of authoring them and increase their usefulness.

However, the fundamental challenge remains – it is both time and effort intensive for a developer to document their decisions with little immediate payoff. Not only is the payoff in the future (which discounts its value [27]), it is also largely for the benefit of others, with uncertainty about whether it will be valuable, relevant, or even comprehensible. Unakite addresses this challenge for at least a certain class of decisions by leveraging the initial developers' decision making process: as developers use the web to search for information, they are incentivized to use a tool that helps them easily keep track of the trade-offs between various options, thereby externalizing their thought processes [22] into a structured form that is useful for decision-making and serves as documentation for later use. Unakite does not currently address the issue of comments going stale, but even if they do, we feel it may be useful for developers to at least know the original design rationale.

There is a recent trend in the research community of trying to automate the generation of various kinds of documentation, such as rationales, commit messages, and release notes [44, 21, 48]. Alkadhi et al. proposed to automatically extract rationale elements by analyzing IRC messages of development teams [13]. However, these approaches rely heavily on the analysis of existing example documentation and often suffer in accuracy and quality. In contrast, Unakite encourages developers to voluntarily keep track of their decision-making processes, which provides more accurate and organized documentation.

Making Sense of Online Information

To help people better make sense of online information, systems like Hunter Gather [61], Dontcheva et al.'s web summarization tool [23], Google Notebook [3], and CheatSheet [67, 68] enable users to collect snippets of content from web pages and later combine them into a single document for easy access and sharing (in [23]'s case, the snippets are algorithmically summarized by combining user labeling and layout presets). Unakite draws from and builds upon this prior work while taking a different approach in terms of organizing by: 1) giving users the complete control of what and how evidence is collected, organized, and presented; and 2) enabling and encouraging users to quickly structure information about trade-offs as they are searching and browsing.

Kittur et al.'s characterization of the costs and benefits of structuring information during sensemaking processes [36, 37], Chang et al.'s system on highlighting content with force touch on mobile phones [17] and other prior work [32, 46] suggest that interactions for collecting and structuring information while browsing need to be quick, intuitive, and low-cost without taxing users with too much cognitive workload. This insight, in particular, guided the design of Unakite, where we iterated to identify the most natural and lightweight interactions for capturing and structuring snippets, such as selecting text, drawing bounding boxes, and dragging-and-dropping.

FORMATIVE INVESTIGATIONS

To gain deeper insights into the barriers developers are facing about trade-offs, we performed two formative studies.

Interview Study

First, we conducted a series of needs-finding interviews with developers to understand how they currently collect and manage information about trade-offs in programming.

Methodology

Participants were a convenience sample of 15 developers (11 male, 4 female) recruited through social media listings and

mailing lists. To capture a variety of processes, we chose 5 professional software developers, 2 doctoral students, and 8 master students. While we do not claim that this sample is representative of all developers, the interviews were very informative and helped motivate the design of Unakite.

We began by asking how frequently participants made decisions about trade-offs when programming. We then explored how they manage these situations. We asked the participants to provide context by reviewing their browser histories and code bases to cue their recollections while retrospectively describing recent projects or problems. We solicited their workflows, strategies, mental models, frustrations, and needs. Finally, we wrapped up with questions probing their experience with understanding programming decisions made by other developers.

The interviews were conducted either in our research lab or remotely by three of the authors and lasted around 30 minutes each. They were recorded and then transcribed. The first author went through the transcripts and coded them using an open coding approach [18], which include discussions with the research team. Our key findings are presented below.

Results

Making decisions about trade-offs is frequent in programming. Almost all programming tasks described by participants involved some level of decision-making that required them to choose among options. In fact, 13 out of 15 said that they were frequently swamped with exploring multiple possible options while trying to compare them based on various criteria, such as the trade-offs among optimization methods when training neural nets (e.g., *"stochastic gradient descent"*, *"augmented Lagrangian"*, etc.) (P9) and the balance between cost and performance when *picking cryptographic algorithms* to protect users' sensitive information (P13).

Participants' browsing patterns and mental models for capturing trade-offs evolve as they dig deeper into the decision space, with a common representation being a comparison table. When approaching decision-making problems like *picking a JavaScript framework to build a web application* (P10), developers generally expected to find a quick-fix style solution at the beginning of their searching process. At this stage, they tended to only curate a short list of solutions that fitted their initial constraints as they queued each in a different browser tab for later reference, without pondering much about the advantages and disadvantages of each. As they dug deeper into the decision space (sometimes voluntarily doing due diligence to investigate multiple options before committing to something permanent (P4, P7), and other times because the previous solution they tried failed), they started to discover new options, criteria, and trade-offs that they were unaware of before. This naturally prompted them to go back to their earlier findings and make comparisons. As reported, their mental models at this stage quickly evolved into a comparison table, with its entries being filled according to information about whether an option satisfied a particular criterion. These findings prompted us to further analyze the applicability of tabular formats in synthesizing the trade-offs in programming problems, which we discuss in the next section.

No matter how organized their tabular mental models might become in the end, most participants reported that their exploration was inherently non-linear and tangled – there was no set pattern that was followed to acquire all the relevant information they needed. For example, as they went through web pages, they discovered new evidence, which in turn drove them to search for or go back to a previous page to read in detail about another option or criterion that they previously missed. This back-and-forth sensemaking process becomes particularly challenging, as evidence is often spread across different web pages on different browser tabs, each with different formats and structures. Additionally, participants often do not realize that there are various trade-offs between options until they dig deeper into the decision space, at which point they are already overloaded with information and lost in browser tabs, and it is hard for them to recall, search for, or go back to previously missed content to fill in the blanks in their mental table. These findings prompted us to offer various features in Unakite to help developers go back to previously visited content such as automatically keeping track of the source URL and the scroll position when collecting information.

Both making decisions and understanding them later are difficult and cognitively demanding, and developers expressed a strong desire for tool support. 8 out of 15 said they used general-purpose tools and methods like taking notes in Google Docs or using a web clipper (such as that provided by Evernote) and reported problems such as: a high cost associated with collecting content (P7: *"...copy-pasting is just too much work, and I lose all the styling; while Evernote clipper clips the entire page, which is equivalent to not saving anything at all [because] I'd have to re-find it later."*); maintaining provenance (P15: *"...whenever I save something, I always forget to also save the URL [of the source]."*); synthesizing the new with existing content (P9: *"Evernote dumps everything I clip into a list of notes. There's no way for me to organize them."*); and guiding their exploration processes (P1: *"... sometimes there's just so much [evidence to find] that I often don't have a clue about what I'm supposed to search next."*). Additionally, participants reported that another disadvantage of using Google Docs or other applications like Evernote is that they must switch to another browser tab or application to access and organize their collected information. Such frequent context switches are tedious and have been shown to harm developers' productivity [29, 34, 47]. These findings inspired us to help developers easily externalize their mental models when they are searching and browsing, by providing an easier method of tracking and deciding among available options.

Almost half (7/15) of the participants admitted that they do not document their decisions anywhere. An additional three said that they would only record important source URLs in code comments. Interestingly, participants also discussed the difficulties in code comprehension, particularly when trying to understand code written by others that involved unexpected decisions. They attributed the frustrations primarily to being unable to uncover the context of the decisions and the original trade-offs, and fearing they might accidentally violate important yet hidden constraints that guided the original decision, which is congruent with prior research [38, 39]. This

motivated Unakite to automatically keep track of the initial developer's decision making trails as the design rationale, unlike prior work where developers are forced to manually create documentation of decisions after they are made [51, 66].

Analysis of Stack Overflow

Stack Overflow (SO) is an important tool for answering programming questions, and participants cited it as their most frequently visited resource. Given this motivation, we undertook an analysis to assess the proportion of questions on SO which capture trade-offs among multiple options and to determine if the tabular format identified in the interviews is indeed an appropriate structure for synthesizing these trade-offs.

We utilized two sets of posts for this analysis. First, we queried the 50 most viewed questions. We were concerned about this sampling method as it may only represent a narrow set of topics which happen to be the most popular, whereas the average developer may have more niche interests [14]. To obtain a sample of questions with a variety of topics that may be more representative of the interests of the general population, we collected another 90 questions by querying for posts created on a particular day which contained three or more answers. Through manual analysis and construction of comparison tables using spreadsheets, we found that the trade-offs contained in 88% of the 50 most-viewed and 49% of the 90 general population questions along with their answers could be reasonably organized into tables. In fact, we found that some answers already included tables to summarize the trade-offs among the options, e.g., [1, 2]. Together with the results from the interviews, these findings motivated the design of Unakite's organization features that let users synthesize information about trade-offs into comparison tables.

Summary of Design Goals

Led by our formative studies findings and prior research, we hypothesize that an effective interface for decision making about trade-offs while searching and browsing should support:

- **Scaffolding:** helping developers form systematic models when approaching decision making problems with trade-offs.
- **Lightweight interactions:** reducing the cost of collecting and organizing content so that the entry barriers for developers to use the tool are low.
- **Summarization:** helping developers synthesize and summarize different pieces of content together and manage them, as suggested by prior work [73, 72, 49].
- **Contextualization:** enabling developers to recreate the context from which information snippets were collected and copied for better sensemaking [53, 62, 58].

UNAKITE

Guided by the design goals above, Unakite enables developers (both experienced and novice) to easily collect any content from any web page into *snippets* (pieces of information) and organize them by *options*, *criteria*, and *evidence* as they are searching and browsing the web, and thereby keep track of their decision-making trails for later reference. Unakite is an extension to the Chrome Web browser and a web application.

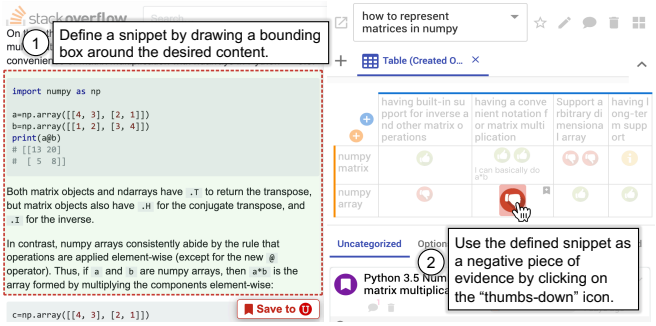

Figure 2: "Teleporting" content directly into the comparison table as a piece of evidence.

We first illustrate the experience of using Unakite by describing a sample usage scenario that embodies many of the use cases identified in our formative studies.

The Unakite User Experience

Sara, a junior professional developer, is tasked with writing Python code to handle matrix calculations for her company. As the code will be used in production, she wants to determine the best way to represent matrices using `numpy` [4] before starting the implementation. She decides to use Unakite to help her stay organized during her exploration process.

Sara logs into Unakite, enables it on her current web pages, brings out the Unakite sidebar (Figure 1-e), and selects "Create a new task", entering "how to represent matrices in `numpy`" as the task name. Next, she starts a Google search on this topic.

As she goes through the search results, she comes across an SO page about the differences between `numpy matrix` and `numpy array`. She then quickly collects text describing both numpy matrix and numpy array into the task snippet repository by just selecting the text and click the "Save to U" button that pops up (Figure 1-a1). The collected snippets immediately appear under the "Uncategorized" tab (Figure 1-c).

Continuing on, she comes across several criteria that seem to be good standards to evaluate which of the two options she just discovered is better. For example, she thinks that "having a convenient notation for matrix multiplication like a*b" is essential for the readability of the code. Therefore, Sara collects those criteria using the same mechanism.

As the number of collected snippets gets larger, Sara decides to quickly organize them by simply dragging and dropping each snippet into the comparison table (automatically created along with the task) above the snippet repository in the sidebar (Figure 1-b). For example, she drags `numpy matrix` into one of the row headers as an option (e.g., a possible solution to solve the task). After a basic table structure is laid out, she realizes that an optimal method should not be deprecated in the future, so she clicks the blue "plus" button to create a new column and types in "having long-term support" as a new criterion. As it's not one of her immediate concerns, she drags that column to be the last one in the table.

To save a section of the SO page that compares the two options in terms of the criteria she just collected, Sara uses the *snapshot* feature (holding the `Option` / `Alt` key and using the

mouse to drag on screen) to draw a bounding box around that section (Figure 1-a2). Instead of clicking the "Save to U" button to save it as a snippet and then drag it into the table (which she certainly can), Sara uses the *teleport* feature (Figure 2) by clicking on one of the rating icons in the corresponding table cells to directly save the snapshot as a snippet and use it as a piece of evidence. For example, she gives `numpy matrix` a "thumbs-up" (positive rating) for "having a convenient notation for matrix multiplication like a*b" and `numpy array` a "thumbs-down" (negative rating) for "having built-in support for inverse and other matrix operations". Alternatively, developers could also label a snippet as "informational" if it does not have a positive or negative effect on their decision (Figure 1-f1,f2,f3).

After filling up the table with options, criteria, and ratings (evidence), Sara now feels clear that `numpy matrix` should be the better choice, so she clicks the green "Choose this option" button (Figure 1-b1) next to that option to indicate it was chosen. She wants to document her decision in the company's internal documentation site. The table she organized, along with all the information snippets she collected, is automatically preserved by Unakite for the current task. She clicks the "Open task detail page" button to open the task in the Unakite dashboard web app, copies the URL from the address bar, and pastes it into her code documentation with "Here's how I decided to choose `numpy matrix`".

A year later, Larry comes in and reads the code along with the Unakite table that Sara created. He glances the ratings and checks the evidence snippets by mousing over the rating icons. He quickly understands Sara's decision, and realizes the opportunity to switch to using a `numpy array` since now the code needs to be able to perform vector operations in arbitrary dimensions and be supported in the long term, both of which are criteria that Sara identified previously.

Detailed Design

We now discuss how the different features in Unakite support the design goals listed previously.

Scaffolding

Unakite provides developers with scaffolding when managing decision making tasks that involve trade-offs by offering the "Option-Criterion-Evidence" (OCE) framework as illustrated in the example scenario. A user can create as many tasks as desired, where typically each task represents a different decision. For each task, the information is organized in a tabular format (Figure 1-b) where options are the row headers, criteria are the column headers, and pieces of evidence are spread across the rest of the cells.

We provide this framework for several reasons. As mentioned in the interview study results, developers' mental model for capturing trade-offs is similar, but less organized, to that described in this framework. Formalizing it provides a concrete framing for developers to think about decisions in a structured way that they are already familiar with. Another aim of providing this structured framework is to encourage developers to think about trade-offs from the start to avoid the unnecessary frustrations later on (as described in the interview results).

Lightweight Interactions

Unakite offers various lightweight interactions to collect information and organize them according to the OCE framework. It provides two intuitive ways to collect any content from any web page. The first is selecting the desired content using the cursor in the normal way, and then clicking the "Save to U" button that pops up (Figure 1-a1). Another way to collect large pieces of information (code snippets that span multiple lines, columns or sub-sections of tables, pictures, etc.) is to use the snapshot feature: drawing a bounding box around the desired content (Figure 1-a2 and Figure 2) and clicking the "Save to U" button. These interactions are carefully designed based on developers' natural habits of copying-and-pasting content and links and taking screenshots without introducing an extra cognitive load of learning a new interaction, and thereby reducing the starting cost for developers to use Unakite.

Unlike previous tools where information was saved either in pure text format [37, 36] or as raw HTML without CSS styling [71], Unakite combines the best of both copying-and-pasting and taking screenshots by capturing, saving and later showing the content of a snippet with its original styling and including the rich, interactive multimedia objects supported by HTML, like images and links. This feature makes the content in snippets more understandable and useful, and also helps developers quickly recognize a particular snippet among many others in the repository by its appearance. Typically, developers will include example code in the snippets as copied from SO and other sources, and Unakite is careful to preserve the formatting of the code, so it can later be copy-and-pasted into the user's code once a decision to use it has been made.

The collected snippets will be displayed in the current tasks' snippet repository (Figure 1-c), which serves as a container that holds all the collected snippets in the form of snippet cards (Figure 1-d). One of the benefits of having this repository is that it serves as an information buffer between the web and the comparison table: as recommended by Kittur et al. [37], a "two-stage" model in which information is first saved and then organized, results in a better "structured information space".

To solve the problem of frequent context switches (identified in the interview study), Unakite brings the ability to access and organize collected information directly into the browser tab that the developer is currently using – Unakite provides a sidebar (inspired by [67, 68]) on the right side of the current window (Figure 1-e) containing the comparison table (Figure 1-b) and the aforementioned snippet repository. There are several major advantages for developers using the Unakite sidebar. It serves as a comprehensive dashboard that contains both the collected information and the ability to organize them into comparison tables (discussed later in detail) all in a small footprint. Unlike PlayByPlay [71] in which the sidebar lives in a part of the browser UI, Unakite's sidebar is directly injected into the DOM tree and therefore can provide rich interactions with the original web page. The sidebar can be toggled in and out like a drawer using the keyboard shortcut `Ctrl + ` ` (backtick) or using the "Open/Close Unakite Sidebar" button on the bottom right of the window. When it opens, it automat-

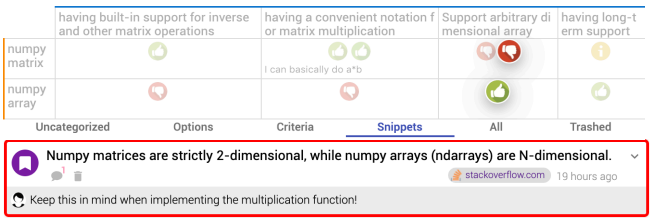

Figure 3: A snippet used as evidence in multiple table cells. Selecting a snippet will highlight its location(s) in the table.

ically shrinks the width of the web page body to make sure nothing is visually hidden.

Unakite provides easy and intuitive interactions such as drag-and-drop, allowing users a variety of ways to quickly organize the collected information into a comparison table. A developer can drag a snippet card from the snippet repository and drop it into the table as either a row header (so it is an option), a column header (as a criterion), or into a cell as a piece of evidence, just as Sara did. Inspired by prior work [49], one can "rate" a snippet as either a positive (shown as a "thumbs-up" rating icon, see Figure 1-f1), negative (shown as a "thumbs-down" rating icon, see Figure 1-f2), or informational (shown as an "info" rating icon, see Figure 1-f3) piece of evidence. Moreover, a snippet can be reused as the evidence in multiple cells. Selecting a snippet (by clicking on it, see Figure 3) in the snippet repository will reveal its location(s) in the comparison table, and selecting an icon in the table opens the corresponding snippet in the repository.

There are two additional shortcuts to put snippets directly into a table. To collect some content as an option or a criterion, one can mouse over the "Save to U" button and click the "Option" or the "Criterion" button (Figure 4) that appears below. This is modeled after the various options for "liking" in Facebook. In addition to collecting the desired content as a snippet, this will automatically create a new row or column in the comparison table. Another shortcut is the teleport feature that Sara used above (Figure 2). These shortcuts are enabled by and add additional benefits to Unakite's always-available sidebar. Together with the other features described above, users have the flexibility to capture and organize their knowledge in various ways and in any order without needing to follow a preset process.

As illustrated in the example scenario, every Unakite task, including all of its snippets and comparison tables, can be accessed in the Unakite web app via a unique URL in any browser. This makes sharing and keeping track of one's decision easier and more powerful: developers can choose to share the link to a task via email to their friends and colleagues to show how and why the decision was made, and the link can be embedded in documentation or comments in code, preserving the actual trade-offs and design rationale in addition to where any example code was copied from.

Summarization

Unakite introduces several levels of summarization to help developers manage and digest information.

The comparison table provides a high-level summary of the decision making space and the trade-offs among various options. It offers a clear and glanceable picture of the advantages and

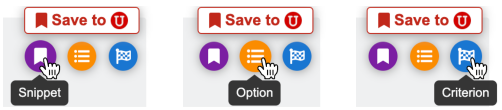

Figure 4: Mousing over the "Save to U" clip button will reveal three additional buttons to collect the desired content specifically as a snippet, an option, or a criterion.

disadvantages of each option through the "thumbs-up" and "thumbs-down" rating icons without having to expose the nitty-gritty details of the evidence content, which is useful both for the developer making the decision and later code readers, as shown in the example scenario. Additionally, it serves as a presentation of one's exploration progress that helps users understand which part of the decision space has been explored and which has not (revealed in the interview studies as an important clue developers need when exploring multiple options). For example, the empty cells in the table provide developers with clues about where they need to focus next.

The individual rating icons provide another level of summary of their corresponding supporting evidence. Unlike in previous summarization tools [73] where contents are recursively summarized into words, Unakite encourages the user to parse out the information in a snippet that captures the relationship between an option and a criterion, and represent them as rating icons. We believe this mechanism can usually capture developers' information needs of whether an option satisfies a specific criterion, as identified in the formative interviews. One can also manually add a rating leveraging their prior knowledge directly in the table by clicking the "Add a snippet" button on the top right of the table cells, and just type or paste. To dig into the detailed evidence of each rating, users can simply click on those icons in the sidebar tables or mouse over the icons in the Unakite web app to reveal the supporting snippet card.

In addition to the built-in summarization mechanisms above, Unakite also enables users to note down their own summaries in various places. Users can easily edit the snippet title (Figure 1-d1) in the snippet card to be something more summative. For example, for a long snippet that talks about the performance advantages of React [24] over Angular [30], a user may summarize it as "React apps load faster than Angular ones." There is also a text box in each table cell for users to summarize all the evidence in that cell or keep track of the evidence that cannot easily be captured by rating icons, such as prices and speed. Moreover, one can add comments to snippets (Figure 1-d4), table cells, and tasks about their opinions, thoughts, or the results of their experiments with an option, etc. These were added based on feedback that developers needed more flexibility to add comments and content in many places.

Contextualization

Meta information such as the URL of the source web page (Figure 1-d2) and the time of collection (Figure 1-d3) are automatically recorded along with the snippet and displayed on the snippet card in Unakite. Using this feature, developers are able to go back to the web page where a snippet was collected. Unakite will even help developers to go back to the exact scroll position where the snippet was collected if possible, saving the extra effort of locating it on a web page. The time when a snippet was collected is especially useful in giving

	# manually created snippets / # snippets	# options	# criteria	# ratings	# positive ratings	# negative ratings	# info ratings
Task 1	0.70 (1.34) / 12.10 (3.38)	2.30 (0.67)	2.70 (1.57)	8.80 (4.10)	3.00 (1.89)	1.80 (2.30)	4.00 (3.80)
Task 2	1.20 (3.16) / 17.50 (4.48)	2.60 (0.52)	4.60 (2.07)	13.20 (4.42)	7.70 (4.08)	2.60 (2.46)	2.90 (2.42)
Task 3	2.00 (3.77) / 18.89 (8.31)	3.74 (1.37)	4.74 (2.58)	12.58 (8.87)	6.37 (5.24)	3.84 (4.29)	2.37 (2.52)

Table 1: Statistics for various Unakite feature usages in Study 1. Statistics are presented in the form of mean (standard deviation) in the table.

developers a rough estimate of the age of the information and helping them determine whether it is still valid (e.g., API methods might be deprecated or trade-offs might change in newer library versions).

Implementation

Both Unakite's Chrome browser extension and the web application are implemented in HTML, JavaScript, and CSS, using the React JavaScript library [24]. Unakite utilizes Google's Firebase for hosting, user authentication and data persistence.

EVALUATION

We conducted two initial user studies of the Unakite system in order to answer the following questions:

- Can developers collect and organize information using Unakite?
- How does Unakite compare to currently available tools like Google Docs?
- Do Unakite tables offer value over just reading through web pages when trying to understand the design rationale?
- How can the design of Unakite be improved?

Both studies are approved by our institution's IRB office.

Study 1 - Authoring Unakite Tables

We carried out a study to evaluate developers' ability to use Unakite to collect and organize information about trade-offs.

Procedure

We recruited 20 participants (15 male, 5 female) aged 23-37 (μ = 26.75, σ = 3.49) from a local research participation pool. The participants were required to be 18 or older, to be fluent in English, and to be experienced in programming. Participants had on average 8.8 years of programming experience, with the longest being around 15 years. 13 participants had professional programming experience, with the rest having experience in college.

In this study, participants were first presented with two tasks each: *(A) how to invoke a function in JavaScript* and *(B) how to create or update a resource using REST APIs*. For each task, they started from scratch without using any information snippets from previous tasks. The study was a between-subjects design, where participants were randomly assigned to either the Unakite or the control condition. In the Unakite condition, participants were given a static web page adapted from a real Stack Overflow page discussing the task topic in each task. Participants were asked to use Unakite to collect and organize information from that single page into a comparison table, and were instructed to inform the researcher when they thought they had finished the task or felt like they could make no further progress. In the control condition, participants were asked to do the same but to build comparison tables using Google Docs instead. We deemed Google Docs as a proper baseline since: 1) it was reported in the formative study as a common tool people use to take notes while making decisions; 2) all participants in this user study were already proficient in using it; 3) compared to other solutions like spreadsheets, it can be easily used to capture richer contexts such as formatted text (example code), images (screenshots of execution results), and links (URLs of documentation and tutorial pages).

All participants were then given a third task in which they were asked to use Unakite to help them understand the trade-offs and make decisions on whatever programming problems they were trying to solve in real life.

Participants in the Unakite condition were given a 10-minute tutorial showcasing the various features of Unakite and a 5-minute practice session before starting. Those in the control condition were given the same tutorial and practice session before the third task. At the end of the study, the researcher conducted a survey and an interview eliciting subjective feedback on the Unakite experience. In particular, participants were asked to list 3 of their favorite features as well as 3 least favorite features or possible improvements of Unakite. The study took about 80 minutes per participant, using a designated MacBook Pro computer with Chrome and Unakite installed. All tasks were screen-recorded for later analysis. All participants were compensated $20 for their time.

Results

All participants were able to complete all of the tasks in both conditions. As shown by the statistics in Table 1, the Unakite participants were able to use the various features to collect and organize information into comparison tables.

To examine how Unakite performs compared to the control condition, we opted to compare the *overhead cost* of using both tools to collect and organize information. For the Unakite condition, the overhead cost is defined as the portion of the time participants spent on directly using Unakite features (selecting, snapshotting, dragging snippets into the comparison table, etc.) out of the total time they used for a task, since the rest of the time was spent reading and understanding the Stack Overflow page. Similarly, for the control condition, the overhead cost was calculated as the percent of time participants spent on copy-and-pasting content, making screenshots, and staying on the Google Docs browser tab to organize the table.

We conducted a mixed-effect linear regression with overhead cost as the outcome, condition, task, and their interaction as fixed effects. Since participants may have different abilities in performing the tasks, we included a random intercept for each participant. Results show that the overhead cost when using Unakite is significantly lower (coefficient $= -0.22$, $t(18) = -4.81$, $p = 0.0001$) than the control condition, while task (coefficient $= -0.05$, $t(18) = -1.40$, $p = 0.1777$) and the interaction term (coefficient $= 0.04$, $t(18) = 0.71$, $p = 0.4861$) does not have an effect on the overhead cost. Across both tasks, the average overhead cost was reduced by 45% when using

Figure 5: Participant P13's comparison table capturing the trade-offs in choosing JavaScript front-end frameworks.

Unakite (Mean overhead cost = 25%, SD = 0.07) compared to using Google Docs (Mean = 44%, SD = 0.12). Thus, using Google Docs did add a lot of extra time, whereas using Unakite, even though unfamiliar, was quick and non-disruptive.

In the survey, participants reported (in 7-point Likert scales) that they thought the interactions with Unakite were understandable and clear (Mean = 6.20, Median = 6.00, 95% CIs = [5.84, 6.56]), they enjoyed Unakite's features (Mean = 6.00, Median = 6.00, 95% CIs = [5.52, 6.48]), and would recommend Unakite to friends and colleagues doing programming work (Mean = 6.20, Median = 6.50, 95% CIs = [5.75, 6.65]).

Nine of the 20 participants requested that we send them the URL of their third task that they created using Unakite for reference and five of them asked us to help them install Unakite on their computer for personal use and future updates, highlighting both the utility of the system as well as the realism of the tasks they chose. Figure 5 shows P13's table capturing the trade-offs in choosing JavaScript front-end frameworks.

Another highlight in the study is that P3, P10, and P18 decided to either commit or switch to the option they identified as the best option after using Unakite to build comparison tables on the topic of their choosing. For example, P3 researched on hybrid AR development frameworks that can take advantage of both ARCore [5] on Android and ARKit [6] on iOS, and found ViroReact [12] to be the best choice. A quick follow-up interview a week later revealed that he had already begun using that framework, and it did satisfy all of his needs so far.

Study 2 - Understanding Unakite Tables

We carried out a second study to evaluate whether developers could understand the trade-offs encapsulated in comparison tables and snippets previously built by others using Unakite.

Procedure

We recruited 16 participants (9 male, 7 female) aged 21-32 (μ = 25.3, σ = 3.19) from the same local participation pool as in Study 1 (but no-one participated in both studies). Participants had on average 7.8 years of programming experience, with the longest being 17 years. None of them were familiar with either the topics involved in this study or Unakite. The study took about 40 minutes per participant, using the same setup as in Study 1. All participants were compensated $15.

Participants were given a 10-minute tutorial showcasing the various features of the Unakite web app. The study was a within-subjects design, where the participants were presented with two tasks of roughly equal difficulty and were asked to solve one of them with the help of Unakite and the other by reading through a set of web pages, in a counterbalanced order. For each task, participants were given some code written by the researcher to solve a problem, some necessary background

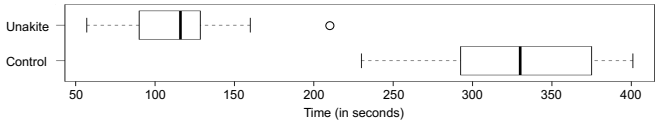

Figure 6: Box plot of the average task completion time for the participants under different conditions: Unakite vs. Control in Study 2.

information about the problem, and a list of options that were available to solve it. They were then asked to explain why the decision was made to choose the particular option used in the code and the associated trade-offs. In the experimental condition, participants were provided with a previously-built structure (including the comparison table and the snippet repository) through the Unakite web app, while in the control condition, participants were instructed to only read through the set of web pages that the structure in the experimental condition was built from. Specifically, the two tasks were to explain the decision and the trade-offs of:

- Choosing `numpy array with Python 3.5+` instead of `numpy matrix` or `numpy array with Python 2.7` to perform some matrix calculations like multiplication, inversion, element-wise multiplication, etc.
- Choosing `numpy array` instead of `Python list` or `Python array` to hold data involved in large-scale numerical manipulations such as regression analysis.

To ensure realism, both tasks were based on actual questions asked and answered on Stack Overflow that are heatedly discussed and well-maintained by real developers.

Results

Two researchers each listed all possible explanations to the two tasks independently. After resolving conflicts, we produced a list of possible explanations for each task as the gold standard. To quantitatively evaluate participants' performance, we measured the time it took for them to offer three legitimate explanations - those within the gold standard list - in each condition, which all participants were able to accomplish.

A two-way repeated measures ANOVA was conducted to examine the within-subject effects of condition (Unakite vs. Control) and task (A vs. B) on task completion time. There was a statistically significant effect of condition ($F(1,26) = 25.59$, $p < .001$) such that participants completed tasks significantly faster (almost 3 times faster) with Unakite (Mean = 114.63s, SD = 38.91s) than in the control condition (Mean = 332.56s, SD = 56.26s), as visualized in Figure 6. There was no significant effect of task ($F(1,26) = 0.01$, $p = 0.94$), indicating the two tasks were indeed of roughly equal difficulty.

Evaluation Discussion

Usability and Usefulness of Unakite's features

The snippet collection features, including both the selecting and the snapshot features, were considered highly useful, with 15 participants citing them as one of their favorite features. Participants said they were *"the perfect combination of copy-pasting and taking screenshots"* (P15) with the additional benefits of *"retaining the original styling [of the collected content], especially when there's code"* (P9), *"keeping track of the [source] URL"* (P7), and *"saving [users] some typing"*

(P5). The drag-and-drop interactions were also popular, receiving 13 mentions in participants' "top three" lists, primarily due to its ease of use (P18: *"it is natural, like picking things up and dropping them in buckets"*). Participants also appreciated that the design of the Unakite UI is clean and easy to learn (12/20), and the overall experience was satisfying (10/20). The sharing via URL feature also received nine mentions, with participants laying out potential usage scenarios like *"putting it in code comments or [their lab's] internal documentions"* (P11), *"using it for presentations in code reviews"* (P8), *"attaching it in emails that explain my code"* (P5), etc.

Compared with using Google Docs, P15 praised the value of Unakite's snippet repository functioning as an information buffer: *"It's like a note-taking space. I can just easily grab as much info that's related to my topic as I want, and they don't have to directly fit into the table, but can be something interesting to use later on; whereas in Google Docs, the cost of buffering these interesting snippets somewhere is pretty high."*

Participants have mixed opinions on how summarization works in Unakite. Most of them (16/20) agreed that summarizing snippets into positive, negative, or informational icons alleviates their burden of having to manually look at the content of each snippet every time, and makes the comparison tables much more skimmable, e.g., *"visual interpretation of thumbs ups and downs provides a quick summary"* (P18). However, P17 also pointed out that *"value comparisons between criteria (columns) are difficult,"* suggesting some notion of weight should be applied differently to the columns when construing the table. P3 indicated that the meaning for the thumbs-up/down icons is open for interpretation in a sense that *"having more thumbs-ups does not necessarily mean [that an option] is better [in terms of a criterion], it could simply mean that the author found more positive evidence, unless she specifies that [more means better] in the first place."* Based on these valuable insights, we believe that there are new interface design opportunities for us to explore in Unakite so that the value of the comparison tables could be further improved.

Usage Patterns

Similar to what Morris et al. found [49], there was an unbalanced use of the positive and negative ratings in the study: positives (228 in total) are more heavily used than negatives (117 in total). A possible explanation for this asymmetry is that people in general lean towards finding and keeping track of evidence of what "works" rather than what "doesn't work".

Participants exhibited two major usage patterns when interacting with Unakite: (1) collecting-oriented: alternating between *long* collecting stages (in which they keep collecting content into the snippet repository) and *short* organizing stages (in which they focus on putting the collected snippets into the comparison table); or (2) organizing-oriented: all snippets going directly into the comparison table immediately after they are collected. We are delighted that interactions in Unakite are flexible enough to support both usage patterns equally well.

The studies showed some evidence that Unakite might also be used for other tasks like comparison shopping for electronics or makeup, even though they are not the focus of Unakite.

CONCLUSION AND FUTURE WORK

Through designing and evaluating Unakite, we gained deeper insights into people's frustrations and needs towards making sense of programming trade-offs on the web. This could pave a path for future work.

Seven of the participants (Study 1 & 2 combined) who were involved in decision making processes in the industry suggested that Unakite has the potential to become a collaborative platform for developers to cooperate on decision making processes. This is in line with our vision to add support in Unakite for both asynchronous and synchronous collaborations in the future. Presently, Unakite focuses on recording a static snapshot of a single developer's decision making trails that is read-only to other developers. In future iterations, we would like to work on mechanisms that enable later developers to "own" or "contribute" to the structures so that they stay relevant and informative throughout the course of a software engineering project. For example, inspired by Git and other local version control tools such as Variolite [35], we can explore the opportunity of introducing lightweight versioning into Unakite, possibly integrated with code versions, thereby realizing asynchronous collaborations. Suggested by collaborative systems like SearchTogether [49] and CoSense [53], additional "awareness" and "division of labor" features can be implemented to transform Unakite into a synchronous collaboration platform.

To support cases in which the needs for collecting and organizing information are not discovered until partway through an investigation process, we will also explore automatically summarizing exploration paths in the background so that developers can retroactively organize their work with reduced overhead.

We also plan to investigate the use of Unakite as a pedagogical tool. Many areas of computer science (e.g., data structures, systems) require students to consider different options in terms of trade-offs, rather than determining a single correct answer. Anecdotally, many students find this difficult. The exercise of creating a comparison table to explicitly compare multiple options for a task (e.g., using a stack or a queue to build an undo function) would force students to explicitly determine the criteria necessary for the task, gather evidence to support ratings, and make an educated decision based on these ratings.

Several participants mentioned in the interviews that Unakite's is *"useful in terms of helping [them] form mental models"* (P4) while searching, especially when there are a lot of equally plausible choices involved. However, P15 also pointed out that the table structure is *" a double-edged sword"* in a sense that it promotes structured thinking but also *"forces [users] to follow a fixed pattern."* In light of these mixed opinions, we would like to leverage the Unakite platform to conduct a long-term field study with two specific goals in mind: (1) exploring the possibility of making the current version of Unakite an intervention mechanism to promote a structured way of approaching decisions about trade-offs and help developers form the habit of staying organized; and (2) exploring different schema of knowledge representation other than tables such as decision trees that could also support developers' decision making about trade-offs and beyond.

ACKNOWLEDGMENTS

This research was supported in part by NSF grant CCF-1814826, Google, Bosch, and the CMU Center for Knowledge Acceleration. Any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the sponsors. We would like to thank our study participants for their kind participation and our anonymous reviewers for their insightful feedback. We are grateful to Felicia Y. Ng and Xu Wang for their assistance on analyzing the study results, and Dr. Walter S. Lasecki, Fanglin Chen, Toby Jia-Jun Li, Haojian Jin, Joseph Chee Chang, Yunpeng Song, Yasha Iravantchi, Shiyan Yan, and Siying Feng for their valuable feedback and constant support.

REFERENCES

[1] 2009a. PUT vs. POST in REST. (2009). https://stackoverflow.com/a/32524385

[2] 2009b. Which equals operator (== vs ===) should be used in JavaScript comparisons? (2009). https://stackoverflow.com/a/26923895

[3] 2012. Google Notebook. (2012). https://www.google.com/googlenotebook/faq.html

[4] 2018. *NumPy — NumPy*. http://www.numpy.org/

[5] 2019a. ARCore - Google Developer | ARCore. (2019). https://developers.google.com/ar/

[6] 2019b. ARKit - Apple Developer. (2019). https://developer.apple.com/arkit/

[7] 2019c. Front-end JavaScript frameworks. (2019). https://github.com/collections/front-end-javascript-frameworks

[8] 2019d. Getting started with machine learning. (2019). https://github.com/collections/machine-learning

[9] 2019e. Programming languages. (2019). https://github.com/collections/programming-languages

[10] 2019f. PyTorch. (2019). https://www.pytorch.org

[11] 2019g. TensorFlow. (2019). https://www.tensorflow.org/

[12] 2019h. ViroReact. (2019). https://viromedia.com/viroreact

[13] R. Alkadhi, M. Nonnenmacher, E. Guzman, and B. Bruegge. 2018. How do developers discuss rationale?. In *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 357–369. DOI: http://dx.doi.org/10.1109/SANER.2018.8330223

[14] Michael S. Bernstein, Jaime Teevan, Susan Dumais, Daniel Liebling, and Eric Horvitz. 2012. Direct Answers for Search Queries in the Long Tail. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12)*. ACM, New York, NY, USA, 237–246. DOI: http://dx.doi.org/10.1145/2207676.2207710

[15] Joel Brandt, Mira Dontcheva, Marcos Weskamp, and Scott R. Klemmer. 2010. Example-centric Programming: Integrating Web Search into the Development Environment. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '10)*. ACM, New York, NY, USA, 513–522. DOI: http://dx.doi.org/10.1145/1753326.1753402

[16] Joel Brandt, Philip J. Guo, Joel Lewenstein, Mira Dontcheva, and Scott R. Klemmer. 2009. Two Studies of Opportunistic Programming: Interleaving Web Foraging, Learning, and Writing Code. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '09)*. ACM, New York, NY, USA, 1589–1598. DOI: http://dx.doi.org/10.1145/1518701.1518944 event-place: Boston, MA, USA.

[17] Joseph Chee Chang, Nathan Hahn, and Aniket Kittur. 2016. Supporting Mobile Sensemaking Through Intentionally Uncertain Highlighting. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology (UIST '16)*. ACM, New York, NY, USA, 61–68. DOI:http://dx.doi.org/10.1145/2984511.2984538

[18] Kathy Charmaz. 2006. *Constructing Grounded Theory: A Practical Guide through Qualitative Analysis*. SAGE. Google-Books-ID: 2ThdBAAAQBAJ.

[19] Yan Chen, Sang Won Lee, Yin Xie, YiWei Yang, Walter S. Lasecki, and Steve Oney. 2017. Codeon: On-Demand Software Development Assistance. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. ACM, New York, NY, USA, 6220–6231. DOI: http://dx.doi.org/10.1145/3025453.3025972

[20] Yan Chen, Steve Oney, and Walter S. Lasecki. 2016. Towards Providing On-Demand Expert Support for Software Developers. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16)*. ACM, New York, NY, USA, 3192–3203. DOI:http://dx.doi.org/10.1145/2858036.2858512

[21] Luis Fernando Cortés-Coy, Mario Linares-Vásquez, Jairo Aponte, and Denys Poshyvanyk. 2014. On Automatically Generating Commit Messages via Summarization of Source Code Changes. In *Proceedings of the 2014 IEEE 14th International Working Conference on Source Code Analysis and Manipulation (SCAM '14)*. IEEE Computer Society, Washington, DC, USA, 275–284. DOI: http://dx.doi.org/10.1109/SCAM.2014.14

[22] Simon P Davies. 1993. Externalising Information During Coding Activities: Effects of Expertise, Environment and Task. In *Empirical Studies of Programmers: Fifth Workshop*, Curtis R Cook, Jean C Scholtz, and James C Spohrer (Eds.). Ablex Publishing Corporation, Palo Alto, CA, 42–61.

[23] Mira Dontcheva, Steven M. Drucker, Geraldine Wade, David Salesin, and Michael F. Cohen. 2006. Summarizing Personal Web Browsing Sessions. In *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology (UIST '06)*. ACM, New York, NY, USA, 115–124. DOI: http://dx.doi.org/10.1145/1166253.1166273

[24] Facebook. 2018. React - A JavaScript library for building user interfaces. (2018). https://reactjs.org/

[25] Beat Fluri, Michael Wursch, and Harald C Gall. 2007. Do Code and Comments Co-Evolve? On the Relation between Source Code and Comment Changes. In *14th Working Conference on Reverse Engineering (WCRE 2007)*. IEEE, 70–79.

[26] Beat Fluri, Michael Wursch, Emanuel Giger, and Harald C Gall. 2009. Analyzing the co-evolution of comments and source code. *Software Quality Journal* 17, 4 (Dec. 2009), 367–394.

[27] Shane Frederick, George Loewenstein, and Ted O'Donoghue. 2002. Time Discounting and Time Preference: A Critical Review. *Journal of Economic Literature* 40, 2 (June 2002), 351–401. DOI: http://dx.doi.org/10.1257/002205102320161311

[28] Andreas Gizas, Sotiris Christodoulou, and Theodore Papatheodorou. 2012. Comparative Evaluation of Javascript Frameworks. In *Proceedings of the 21st International Conference on World Wide Web (WWW '12 Companion)*. ACM, New York, NY, USA, 513–514. DOI:http://dx.doi.org/10.1145/2187980.2188103

[29] Victor M. González, Gloria Mark, and Gloria Mark. 2004. "Constant, Constant, Multi-tasking Craziness": Managing Multiple Working Spheres. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '04)*. ACM, New York, NY, USA, 113–120. DOI: http://dx.doi.org/10.1145/985692.985707 event-place: Vienna, Austria.

[30] Google. 2019. Angular - One Framework. Mobile & Desktop. (2019). https://angular.io/

[31] Nathan Hahn, Joseph Chee Chang, and Aniket Kittur. 2018. Bento Browser: Complex Mobile Search Without Tabs. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. ACM, Montreal QC, Canada, 251:1–251:12. DOI: http://dx.doi.org/10.1145/3173574.3173825

[32] Ken Hinckley, Xiaojun Bi, Michel Pahud, and Bill Buxton. 2012. Informal Information Gathering Techniques for Active Reading. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12)*. ACM, New York, NY, USA, 1893–1896. DOI: http://dx.doi.org/10.1145/2207676.2208327 event-place: Austin, Texas, USA.

[33] Raphael Hoffmann, James Fogarty, and Daniel S. Weld. 2007. Assieme: Finding and Leveraging Implicit References in a Web Search Interface for Programmers. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology (UIST '07)*. ACM, New York, NY, USA, 13–22. DOI: http://dx.doi.org/10.1145/1294211.1294216 event-place: Newport, Rhode Island, USA.

[34] Mik Kersten and Gail C. Murphy. 2006. Using Task Context to Improve Programmer Productivity. In *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering (SIGSOFT '06/FSE-14)*. ACM, New York, NY, USA, 1–11. DOI:http://dx.doi.org/10.1145/1181775.1181777 event-place: Portland, Oregon, USA.

[35] Mary Beth Kery, Amber Horvath, and Brad Myers. 2017. Variolite: Supporting Exploratory Programming by Data Scientists. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17)*. ACM, New York, NY, USA, 1265–1276. DOI:http://dx.doi.org/10.1145/3025453.3025626

[36] Aniket Kittur, Andrew M. Peters, Abdigani Diriye, and Michael Bove. 2014. Standing on the Schemas of Giants: Socially Augmented Information Foraging. In *Proceedings of the 17th ACM Conference on Computer Supported Cooperative Work & Social Computing (CSCW '14)*. ACM, New York, NY, USA, 999–1010. DOI:http://dx.doi.org/10.1145/2531602.2531644

[37] Aniket Kittur, Andrew M. Peters, Abdigani Diriye, Trupti Telang, and Michael R. Bove. 2013. Costs and Benefits of Structured Information Foraging. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '13)*. ACM, New York, NY, USA, 2989–2998. DOI: http://dx.doi.org/10.1145/2470654.2481415

[38] Andrew J Ko, Robert DeLine, and Gina Venolia. 2007. Information Needs in Collocated Software Development Teams. In *29th International Conference on Software Engineering (ICSE'07)*. IEEE, 344–353.

[39] Thomas D LaToza, David Garlan, James D Herbsleb, and Brad A Myers. 2007. Program comprehension as fact finding. In *ESEC/FSE 2007: ACM SIGSOFT Symposium on the Foundations of Software Engineering*. 361–370.

[40] Thomas D. LaToza and Brad A. Myers. 2010. Hard-to-answer Questions About Code. In *Evaluation and Usability of Programming Languages and Tools (PLATEAU '10)*. ACM, New York, NY, USA, 8:1–8:6. DOI:http://dx.doi.org/10.1145/1937117.1937125

[41] Thomas D. LaToza, Gina Venolia, and Robert DeLine. 2006. Maintaining Mental Models: A Study of Developer Work Habits. In *Proceedings of the 28th International Conference on Software Engineering (ICSE '06)*. ACM, New York, NY, USA, 492–501. DOI: http://dx.doi.org/10.1145/1134285.1134355

[42] John Lawrence, Jonas Malmsten, Andrey Rybka, Daniel Sabol, and Ken Triplin. 2017. Comparing TensorFlow Deep Learning Performance Using CPUs, GPUs, Local PCs and Cloud. *Publications and Research* (May 2017). https://academicworks.cuny.edu/bx_pubs/50

[43] K. Lei, Y. Ma, and Z. Tan. 2014. Performance Comparison and Evaluation of Web Development Technologies in PHP, Python, and Node.js. In *2014 IEEE 17th International Conference on Computational Science and Engineering*. 661–668. DOI: http://dx.doi.org/10.1109/CSE.2014.142

[44] W. Maalej and H. Happel. 2010. Can development work describe itself?. In *2010 7th IEEE Working Conference on Mining Software Repositories (MSR 2010)*. 191–200. DOI:http://dx.doi.org/10.1109/MSR.2010.5463344

[45] Mark Mahoney. 2017. Collaborative Software Development Through Reflection and Storytelling. In *Companion of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing*. ACM, 13–16.

[46] Catherine C. Marshall and Sara Bly. 2005. Saving and Using Encountered Information: Implications for Electronic Periodicals. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '05)*. ACM, New York, NY, USA, 111–120. DOI: http://dx.doi.org/10.1145/1054972.1054989 event-place: Portland, Oregon, USA.

[47] André N. Meyer, Thomas Fritz, Gail C. Murphy, and Thomas Zimmermann. 2014. Software Developers' Perceptions of Productivity. In *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2014)*. ACM, New York, NY, USA, 19–29. DOI: http://dx.doi.org/10.1145/2635868.2635892 event-place: Hong Kong, China.

[48] L. Moreno, G. Bavota, M. D. Penta, R. Oliveto, A. Marcus, and G. Canfora. 2017. ARENA: An Approach for the Automated Generation of Release Notes. *IEEE Transactions on Software Engineering* 43, 2 (Feb. 2017), 106–127. DOI: http://dx.doi.org/10.1109/TSE.2016.2591536

[49] Meredith Ringel Morris and Eric Horvitz. 2007. SearchTogether: An Interface for Collaborative Web Search. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology (UIST '07)*. ACM, New York, NY, USA, 3–12. DOI: http://dx.doi.org/10.1145/1294211.1294215

[50] Stephen Oney and Joel Brandt. 2012. Codelets: Linking Interactive Documentation and Example Code in the Editor. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '12)*. ACM, New York, NY, USA, 2697–2706. DOI: http://dx.doi.org/10.1145/2207676.2208664

[51] Soya Park, Amy X. Zhang, and David R. Karger. 2018. Post-literate Programming: Linking Discussion and Code in Software Development Teams. In *The 31st Annual ACM Symposium on User Interface Software and Technology Adjunct Proceedings (UIST '18 Adjunct)*. ACM, New York, NY, USA, 51–53. DOI: http://dx.doi.org/10.1145/3266037.3266098 event-place: Berlin, Germany.

[52] Priyadarshini Patil, Prashant Narayankar, Narayan D.G., and Meena S.M. 2016. A Comprehensive Evaluation of Cryptographic Algorithms: DES, 3DES, AES, RSA and Blowfish. *Procedia Computer Science* 78 (Jan. 2016), 617–624. DOI: http://dx.doi.org/10.1016/j.procs.2016.02.108

[53] Sharoda A. Paul and Meredith Ringel Morris. 2009. CoSense: Enhancing Sensemaking for Collaborative Web Search. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '09)*. ACM, New York, NY, USA, 1771–1780. DOI: http://dx.doi.org/10.1145/1518701.1518974

[54] Ksenia Peguero, Nan Zhang, and Xiuzhen Cheng. 2018. An Empirical Study of the Framework Impact on the Security of JavaScript Web Applications. In *Companion Proceedings of the The Web Conference 2018 (WWW '18)*. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, Switzerland, 753–758. DOI: http://dx.doi.org/10.1145/3184558.3188736 event-place: Lyon, France.

[55] Luca Ponzanelli, Alberto Bacchelli, and Michele Lanza. 2013. Seahawk: Stack Overflow in the IDE. In *2013 35th International Conference on Software Engineering (ICSE)*. IEEE, San Francisco, CA, USA, 1295–1298. DOI:http://dx.doi.org/10.1109/ICSE.2013.6606701

[56] Luca Ponzanelli, Simone Scalabrino, Gabriele Bavota, Andrea Mocci, Rocco Oliveto, Massimiliano Di Penta, and Michele Lanza. 2017. Supporting Software Developers with a Holistic Recommender System. In *Proceedings of the 39th International Conference on Software Engineering (ICSE '17)*. IEEE Press, Piscataway, NJ, USA, 94–105. DOI: http://dx.doi.org/10.1109/ICSE.2017.17

[57] Paruj Ratanaworabhan, Benjamin Livshits, and Benjamin G. Zorn. 2010. JSMeter: Comparing the Behavior of JavaScript Benchmarks with Real Web Applications. In *Proceedings of the 2010 USENIX Conference on Web Application Development (WebApps'10)*. USENIX Association, Berkeley, CA, USA, 3–3. http://dl.acm.org/citation.cfm?id=1863166.1863169 event-place: Boston, MA.

[58] Daniel M. Russell, Mark J. Stefik, Peter Pirolli, and Stuart K. Card. 1993. The Cost Structure of Sensemaking. In *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems (CHI '93)*. ACM, New York, NY, USA, 269–276. DOI: http://dx.doi.org/10.1145/169059.169209

[59] N. Rutar, C. B. Almazan, and J. S. Foster. 2004. A comparison of bug finding tools for Java. In *15th International Symposium on Software Reliability Engineering*. 245–256. DOI: http://dx.doi.org/10.1109/ISSRE.2004.1

[60] Caitlin Sadowski, Kathryn T. Stolee, and Sebastian Elbaum. 2015. How Developers Search for Code: A Case Study. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2015)*. ACM, New York, NY, USA, 191–201. DOI: http://dx.doi.org/10.1145/2786805.2786855

[61] M. C. schraefel, Yuxiang Zhu, David Modjeska, Daniel Wigdor, and Shengdong Zhao. 2002. Hunter Gatherer: Interaction Support for the Creation and Management of Within-web-page Collections. In *Proceedings of the 11th International Conference on World Wide Web (WWW '02)*. ACM, New York, NY, USA, 172–181. DOI: http://dx.doi.org/10.1145/511446.511469

[62] Johanna Shelby and Robert Capra. 2011. Sensemaking in collaborative exploratory search. *Proceedings of the American Society for Information Science and Technology* 48, 1 (2011), 1–3. DOI: http://dx.doi.org/10.1002/meet.2011.14504801318

[63] Jonathan Sillito, Gail C. Murphy, and Kris De Volder. 2006. Questions Programmers Ask During Software Evolution Tasks. In *Proceedings of the 14th ACM SIGSOFT International Symposium on Foundations of Software Engineering (SIGSOFT '06/FSE-14)*. ACM, New York, NY, USA, 23–34. DOI: http://dx.doi.org/10.1145/1181775.1181779

[64] J Stylos and B A Myers. 2006. Mica: A Web-Search Tool for Finding API Components and Examples. In *Visual Languages and Human-Centric Computing (VL/HCC'06)*. 195–202.

[65] T Tenny. 1988. Program readability: procedures versus comments. *IIEEE Trans. Software Eng.* 14, 9 (1988), 1271–1279.

[66] Michael L Van De Vanter. 2002. The documentary structure of source code. *Information and Software Technology* 44, 13 (Oct. 2002), 767–782.

[67] Laton Vermette, Parmit Chilana, Michael Terry, Adam Fourney, Ben Lafreniere, and Travis Kerr. 2015. CheatSheet: A Contextual Interactive Memory Aid for Web Applications. In *Proceedings of the 41st Graphics Interface Conference (GI '15)*. Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 241–248. http://dl.acm.org/citation.cfm?id=2788890.2788933 event-place: Halifax, Nova Scotia, Canada.

[68] Laton Vermette, Shruti Dembla, April Y. Wang, Joanna McGrenere, and Parmit K. Chilana. 2017. Social CheatSheet: An Interactive Community-Curated Information Overlay for Web Applications. *Proc. ACM Hum.-Comput. Interact.* 1, CSCW (Dec. 2017), 102:1–102:19. DOI:http://dx.doi.org/10.1145/3134737

[69] Ryen W. White, Bill Kules, Steven M. Drucker, and m c schraefel. 2006. Supporting Exploratory Search, Introduction, Special Issue, Communications of the ACM. *Commun. ACM* 49 (April 2006), 36–39. https://eprints.soton.ac.uk/263649/

[70] Doug Wightman, Zi Ye, Joel Brandt, and Roel Vertegaal. 2012. SnipMatch: Using Source Code Context to Enhance Snippet Retrieval and Parameterization. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology (UIST '12)*. ACM, New York, NY, USA, 219–228. DOI: http://dx.doi.org/10.1145/2380116.2380145 event-place: Cambridge, Massachusetts, USA.

[71] Heather Wiltse and Jeffrey Nichols. 2009. PlayByPlay: Collaborative Web Browsing for Desktop and Mobile Devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '09)*. ACM, New York, NY, USA, 1781–1790. DOI: http://dx.doi.org/10.1145/1518701.1518975 event-place: Boston, MA, USA.

[72] Amy X. Zhang and Justin Cranshaw. 2018. Making Sense of Group Chat Through Collaborative Tagging and Summarization. *Proc. ACM Hum.-Comput. Interact.* 2, CSCW (Nov. 2018), 196:1–196:27. DOI: http://dx.doi.org/10.1145/3274465

[73] Amy X. Zhang, Lea Verou, and David Karger. 2017. Wikum: Bridging Discussion Forums and Wikis Using Recursive Summarization. In *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing (CSCW '17)*. ACM, New York, NY, USA, 2082–2096. DOI: http://dx.doi.org/10.1145/2998181.2998235