How Programmers Interact with Multimodal Software Documentation

Deeksha M. Arya
McGill University
Montreal, Canada
deeksha.arya@mail.mcgill.ca

Jin L.C. Guo*
McGill University
Montreal, Canada
jguo@cs.mcgill.ca

Martin P. Robillard*
McGill University
Montreal, Canada
robillard@acm.org

***Abstract*—There is a wide variety of online documentation to learn about a given software technology, and prior research has reported that programmers must invest time and effort to identify one that best suits their need. We evaluated five modalities to present information that enable a software document to cater to the different presentation needs of programmers. We developed a prototype tutorial with these modalities on three topics in Java, namely, regular expressions, inheritance, and exception handling. We investigated how people interact with the modalities in the tutorial given a programming topic and a type of task. We conducted a survey study with 56 respondents and confirm that although text content is most useful for solving conceptual tasks, code examples support deeper comprehension of the underlying concepts. Furthermore, we report that respondents' contradicting preferences for the modalities suggest the need to have multiple alternatives in a software tutorial.**

***Index Terms*—software documentation, information needs, documentation preferences, information seeking, multimodality**

I. Introduction

Programmers rely on documentation to understand and use a software technology. Available software documentation varies in its structure, organization, and content. For example, a document can have multiple elements such as overview information, code snippets, and advanced pages [54] in addition to sections, links, and images [53]. Online programming tutorials do not have a standard for their presentation, and thus come in a variety of styles [7]. However, users find it difficult to navigate such resources and determine where the information they are looking for would be located [38].

Additionally, information seekers have different pre-existing preferences about the content or style of resources [13], [6]. For example, some developers refer to code in order to duplicate it [28], and thus may prefer *complete* code examples that can be executed [8]. Other developers find small code examples focused on patterns of usage useful [49]. Finding a balance between both factors is difficult, as being too concise may lead to the issue of *incompleteness*, and being detailed may lead to difficulties with *readability* due to the verbose content [4]. Thus, creating documentation must account for the audience and their varied preferences.

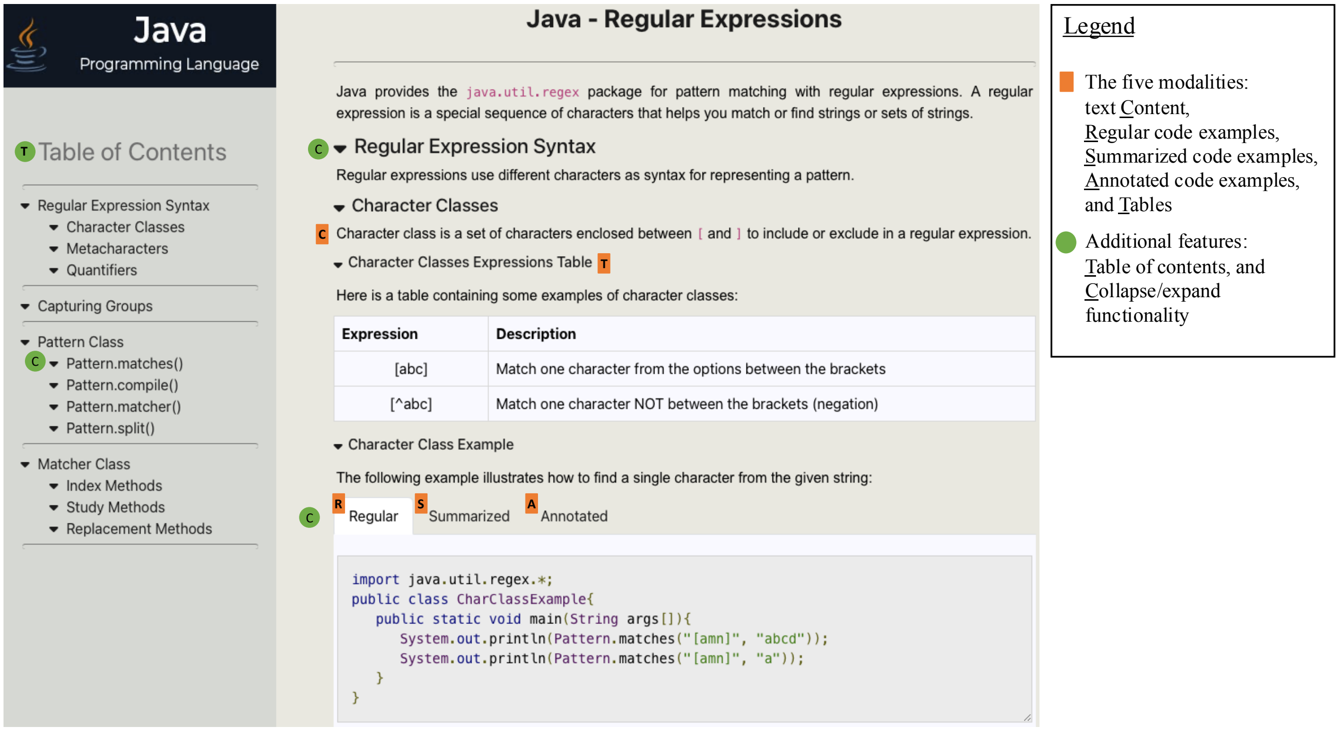
Priestley suggested the idea of a *dynamically assembled document* [43]. Such a document would allow users to restructure the information present into multiple views based on their needs and preferences. Robillard et al. proposed *on-demand developer documentation*, an automatically generated document built based on knowledge of task context and users' needs [50]. In both visions of documentation, a critical aspect is that the presentation and organization of the documentation must cater to users' preferences. But what users need or prefer from documentation can depend on their roles and responsibilities [13]. Whereas it may seem intuitive that these preferences can also vary based on the specific types of tasks they perform, we find little evidence in the literature to support this hypothesis. In this work, we investigated how documentation can cater to the varied presentation needs and preferences of users for different types of programming tasks.

We studied documentation *modalities*, inspired by diverse presentation formats including variations in the conciseness of information. We designed a prototype tutorial containing five modalities, i.e., text, tables, and code examples in three different modalities (regular, summarized, and annotated). We refer to such a document as a *multimodal software tutorial*. A multimodal tutorial presents information through various modalities that allow users to select information according to their presentation preferences. We investigated the research question: *How do programmers make decisions about their presentation needs and preferences in a programming tutorial?* Henceforth, we use the term *documentation* to refer to software documentation in the context of our study, namely, tutorials.

We created three multimodal tutorials using HTML, Javascript, and CSS. The three tutorials are about three basic programming concepts in the Java language: regular expressions, inheritance, and exception handling. We conducted a survey with users that have at least one year of prior programming experience. In the survey, we asked respondents to complete three different programming tasks related to one of three *task types*, i.e., *conceptual*, *how-to*, or *debugging*. After completing each task, we asked respondents to indicate which modalities they used for the task and to explain their choices. We analyzed their responses to determine how they made decisions about information presentation.

We report on how respondents used the different modalities for the different task types. We support our findings with statistical analysis of the responses and insights from open text responses. We observed that, irrespective of the topic, for con-

*Both authors contributed equally to this research.



Java - Regular Expressions

Java provides the `java.util.regex` package for pattern matching with regular expressions. A regular expression is a special sequence of characters that helps you match or find strings or sets of strings.

C Regular Expression Syntax

Regular expressions use different characters as syntax for representing a pattern.

Character Classes

C Character class is a set of characters enclosed between [and] to include or exclude in a regular expression.

Character Classes Expressions Table T

Here is a table containing some examples of character classes:

Expression	Description
[abc]	Match one character from the options between the brackets
[^abc]	Match one character NOT between the brackets (negation)

Character Class Example

The following example illustrates how to find a single character from the given string:

```
import java.util.regex.*;
public class CharClassExample{
   public static void main(String args[]){
      System.out.println(Pattern.matches("[amn]", "abcd"));
      System.out.println(Pattern.matches("[amn]", "a"));
   }
}
```

Legend

🟧 The five modalities:
text Content,
Regular code examples,
Summarized code examples,
Annotated code examples,
and Tables

● Additional features:
Table of contents, and
Collapse/expand
functionality

Fig. 1: Illustration (with excerpts) of a multimodal tutorial for regular expressions in Java. The tutorial prototypes we created for each of the three topics, i.e. regular expressions, inheritance, and exception handling, provide more information through each of the modalities.

ceptual tasks, respondents found textual content "very useful" to complete the tasks, while code examples provided additional context to support comprehension. Similarly, more respondents found regular code examples "very useful" for how-to tasks, and used other modalities for in-depth understanding. Despite these associations, we found that respondents preferred to have access to more than one modality. Respondents also had contradicting preferences. Our findings suggest the need for flexible documentation design that allows users to manipulate the presentation and organization of information content to their needs and preferences, appropriately for different programming contexts.

II. Study Design

We created a prototype multimodal tutorial for each of three Java programming topics. Although a number of documentation types exist [44], we focused on *tutorials*, as they are a commonly used source of software technology information [21]. We conducted a survey to understand whether programmers found the modalities useful and how they used them to complete three types of programming tasks.

A. *Multimodal Tutorial Prototype*

We developed a multimodal prototype tutorial as a static HTML file supported by Javascript and CSS. We provide some of the information in five different *modalities* of information presentation, namely *text content*, *regular code examples*, *summarized code examples*, *annotated code examples*, and *tables*. We provided two additional features, i.e. a *table of contents*, and the ability to *collapse and expand* sections and modalities. Figure 1 shows an illustration of part of a multimodal tutorial. The five modalities (represented by 🟧) and the additional features (represented by ●) are as follows:

C **Text content:** Text-based tutorials are the most common source of programming information [16]. We provided information in textual format, in the form of short paragraphs or bullet points. We followed the twelve filtered guidelines elicited by Miniukovich et al., for designing a readable web page, including *using short, simple sentences in a direct style* and *avoiding complex language and jargon* [39].

R **Regular code examples:** Code examples can contribute to a document's effectiveness [17]. Developers may search for code snippets for reuse in their own use cases [56]. We provided information in regular code examples, i.e. complete code snippets with no additional comments or annotations.

S **Summarized code examples:** Small code examples that show patterns of method usage are reported to be more useful than an example of a single call to the method [49]. We provided *summarized* code examples, by following the selection and presentation practices for summarizing code examples reported by Ying and Robillard [58].

TABLE I: Examples of the three programming task types in our survey.

Task type	**Example task (from survey on regular expressions in Java)**
Conceptual	You are debating whether to use the `matches()` method in the `Pattern` class or the `matches()` method in the `Matcher` class. What is the difference between both these methods?
How-to	The user is asked to input their email address in the expected format: `username@domain.com`. Use regular expressions and write the code to verify that their email address matches the expected format, and then retrieve just the username from their email address.
Debugging	The user is asked to enter their ten-digit phone number which may or may not be separated by hyphens into three parts of 3, 3, and 4 digits (no spaces are allowed). So, valid number formats include: 123-456-7890 and 123-4567890 and 1234567890. You develop this simple regular expression as the pattern in the matches() method: `\d{3}-?\d{3}-?\d{4}` However, when you try to compile your code, the compiler throws an error on this regular expression. What is the issue and how can you fix this regular expression to fit the given criteria?

A **Annotated code examples:** Code examples can benefit from explanations, such as about how the code works or the rationale behind code lines. We provided code examples annotated with additional information that can be revealed on-demand by hovering over selected code elements [40]. We refer to them as *annotated* code examples. We provided the three types of code examples through navigation tabs as shown in Figure 1, with regular code examples in focus by default to imitate a current, common tutorial.

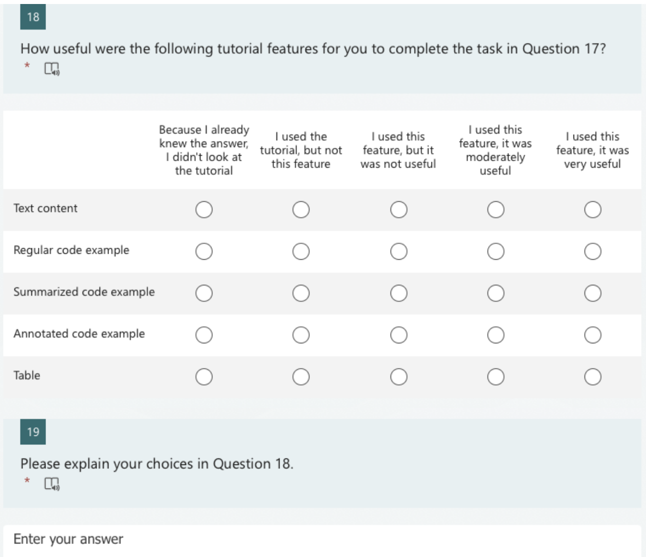
T **Tables:** Tables provide a concise way to present information that captures different types of relations [22]. We presented information in table format to provide an overview of common terms and syntax, and their simplified descriptions.

Additionally, we provided two features to support navigating the contents of the tutorial.

T **Table of contents:** We divided each prototype tutorial page into two containers: the table of contents placed to the left of the page [41], [54], and the main tutorial body occupying the remainder of the page. The table of contents provides an overview of the tutorial, particularly the sections and subsections. A user can navigate to a corresponding section or subsection directly by clicking on the header in the table of contents. This feature allows users to navigate the tutorial's content in a modular manner, in addition to sequentially navigating through the main tutorial body [8].

C **Collapse/expand:** Prior work has suggested that users can benefit from having information revealed to them gradually [41]. We provided a way for users to collapse and expand the five modalities. We designed the three types of code examples, namely *regular*, *summarized*, and *annotated* as alternative navigable tabs, such that one tab could be focused at a time. Additionally, we provided a small clickable black arrow next to a section header, code example title, or table title that allowed users to show or hide the associated content. We also provided the ability to collapse and expand sections via the table of contents.

We leveraged the existing HTML structure of Java tutorials from tutorialspoint.com. We incorporated information from tutorials on other programming knowledge websites such as beginnersbook.com and oracle.com to build a comprehensive tutorial. We built multimodal tutorials about three topics that are basic concepts in the Java programming language: *regular expressions*, *inheritance*, and *exception handling*.



18

How useful were the following tutorial features for you to complete the task in Question 17?
*

	Because I already knew the answer, I didn't look at the tutorial	I used the tutorial, but not this feature	I used this feature, but it was not useful	I used this feature, it was moderately useful	I used this feature, it was very useful
Text content	○	○	○	○	○
Regular code example	○	○	○	○	○
Summarized code example	○	○	○	○	○
Annotated code example	○	○	○	○	○
Table	○	○	○	○	○

19

Please explain your choices in Question 18.
*

Enter your answer

Fig. 2: The follow-up questions to a task that ask respondents for their ratings for the different modalities. Note that the question refers to modalities as "features" (see Section II-B).

B. Survey Design

We created three survey forms, corresponding to each of the three programming topics. Each respondent completed one of these forms, and thus accessed only one of the three multimodal tutorials.

We organized the survey in four parts. The first part collected demographic information from the respondents. In the second part, we required respondents to watch a three-minute video about the associated tutorial, and its modalities and features. We then asked the respondents three control questions to ensure that they had watched the video.

In the third part of the survey, we provided a link to the corresponding multimodal tutorial page, and asked participants

TABLE II: Demographics of survey respondents.

		Age					**Gender**			**Region**			**Prog. Exp. (years)**		
	#Res.	18-24	25-34	35-45	45-54	55-64	Man	Woman	N.S.	N.A.	Asia	Europe	1-5	5-10	>10
Regular Expressions	13	11	1	1	0	0	8	5	0	10	1	2	9	3	1
Inheritance	22	16	4	0	1	1	11	9	2	17	2	3	19	2	1
Exception Handling	21	13	7	1	0	0	14	7	0	20	0	1	12	8	1
Total	**56**	40	12	2	1	1	33	21	2	47	3	6	40	13	3

#Res. — Total number of respondents for that topic
Prog. Exp. (years) — Programming Experience in years,
N.S. — Prefer not to say
N.A. — North America

Q: Did you use the table of contents? If yes, please explain how you used it. If no, please explain why you did not use it.
Q: Did you collapse and expand sections, tables, or code examples? If yes, please explain how these additional tutorial features were useful. If no, please explain why you did not use them.
Q: Were there any other tutorial features you wish the tutorial had?
Q: Please share any additional comments that you have about your experience using the tutorial.

Fig. 3: Optional open-ended questions in the survey.

to complete three programming tasks. The three programming tasks are based on *search intent* categories proposed by Rao et al [47]. Rao et al. identified seven search intents, i.e. reasons for searching for technical information, based on a manual analysis of 400 queries logged by the Bing search engine that are related to Software Engineering. We selected the three intents that are most relevant to basic programming concepts, and thus our context: *Learn*, *How-to*, and *Debug*. From these intents, we defined three programming *task types*, namely *conceptual* (corresponding to the Learn intent),[1] *how-to*, and *debugging*. Table I provides an an example of each task type. We created one programming task corresponding to each of these task types, for each of the three topics in our user study. Thus, each respondent completed a total of three tasks, one of each task type, and all associated with one of the three programming topics. After each task, we asked respondents two follow-up questions: a choice-based question about what modalities of the tutorial they used to complete each task, and an open-ended question to explain their choices (see Figure 2). In the survey, we referred to the modalities also as "features" to have a relatable terminology for respondents.

In the final part of the survey, we asked four open-ended questions about the usefulness of the table of contents and the collapse/expand functionality, any other modalities or features respondents would have liked, and any additional comments they had (see Figure 3).

We conducted six pilot studies, twice for each tutorial. No content-related changes were made to the programming tasks or the tutorials after these pilots; we only fixed typos pointed out during the pilots and refined the demographic questions. The study protocol was approved by the ethics review board of McGill University.

C. Respondent Recruitment

We recruited candidates from student mailing lists within universities, relevant public email groups, software-engineering related Slack channels, and social media channels. We required that interested candidates send an email from their institutional email ID to the first author, who would then respond with a survey link corresponding to one of the three prototypes. The three survey forms were rotated in a round-robin among candidates who contacted us. We received a total of 108 requests to participate in the survey. A total of 70 of these candidates completed the survey, of which four had less than one year of programming experience (the minimum criteria to participate in the survey), seven answered a control question incorrectly, and another responded to all mandatory open-ended answers with "Test". Additionally, two respondents did not respond to our follow-up to clarify their survey responses. The remaining 56 respondents were entered into a draw for a gift card worth CAD $100, with at least a 10% chance of winning.

D. Analysis

Table II shows the demographics of the valid respondents of our survey. We performed tests to determine whether there is a statistically significant association between modalities, their rating, the type of programming task, and the programming topics. We refer to these four as *dimensions* when discussing the statistical analysis. We conducted a total of 16 Fisher's exact tests. We used 200,000 Monte Carlo simulations [37] to account for the multiple categories of each dimension. We also applied a Bonferroni correction by multiplying the p-value[2] obtained from each test by 16, to mitigate Type-I error when making multiple comparisons [1]. We performed 16 Fisher's exact tests between modality rating and each of the other dimensions, using one of the dimensions as a filter, as shown in Table III.

[1] Although the underlying definition is the same, we chose to rename *Learn* to *Conceptual* in our study, because in the context of using software tutorials, all task types may involve some learning. Instead, conceptual specifically refers to learning about a topic, such as comparing two concepts.

[2] We multiplied each p-value to report the "adjusted p-values", as opposed to dividing the alpha value by the number of tests, to support interpretability of the Bonferroni correction and reporting of results [10], [23], [55].

TABLE III: Description of the 16 Fisher's exact tests we performed. We conducted the tests between Dimension A and Dimension B, for each Filter.

Dimension A	Dimension B	Filter	# of tests	Description
Modality rating	Topic	Modality	5 (one for each modality)	These tests indicate whether, for a particular modality, its rating is associated with a programming topic.
Modality rating	Task type	Modality	5 (one for each modality)	These tests indicate whether, for a particular modality, its rating is associated with the type of task.
Modality rating	Modality	Topic	3 (one for each topic)	These tests indicate whether, for a particular topic, there is an association between each modality and the ratings they receive.
Modality rating	Modality	Task type	3 (one for each task type)	These tests indicate whether, for a particular task type, there is an association between each modality and the ratings they receive.

We also calculated the adjusted standardized residuals for each statistically significant association between two dimensions, to determine which pairs of categories across the two participating dimensions have an effect on the association [51]. Residual values greater than +2 indicate a meaningful number of observations more than expected, whereas residual values lesser than -2 indicate that the observations of the pairs of categories are lesser than expected.

For additional insight into the use of tutorial modalities, we analyzed the open-ended text responses of the survey. We open-coded the responses to identify the rationale for using particular modalities. Furthermore, we report on the text responses for the optional questions, including respondents' use of additional features provided in the tutorials.

We report the significant results from our analysis and use the text responses to provide explanations for our observations in Section III. We provide the questions in the survey and the complete statistical analysis results in our online appendix.[3]

E. Study Design Trade-offs

In designing both the multimodal tutorial prototype as well as the survey, we made a number of deliberate decisions that may have impacted the number of respondents and modality rating responses. We discuss the trade-offs of these decisions to communicate our design considerations. [48]. In all three prototypes, we implemented navigation tabs for the code examples in the order *regular*, *summarized*, *annotated*, with the *regular* tab in focus every time the page is loaded. Thus, participants would have to perform additional keystrokes in order to see the *summarized* and *annotated* code examples. As an alternative, we considered allowing respondents to select which of the code examples they would like to see by default, prior to loading the HTML. However, this would have required them to be familiar with all three types of code examples *before* using the tutorial, which was uncertain. To ensure that survey respondents were aware of the other tabs with other modalities, we prepared and provided a three minute tutorial video about all the tutorials modalities in the survey. We used control questions, e.g. *Please select the statement that best describes "annotated code examples" in the video*, to ensure that respondents were aware of all available modalities before using the tutorial for the survey. We did observe that more respondents found regular code examples useful than other types of code examples for all programming tasks and topic. However, respondents acknowledged the usefulness of summarized and annotated code examples based on their own preferences and in different contexts (see Section III-D).

Once deployed, the survey was potentially subject to invalid and false responses. To mitigate the possibility of spam, we required all interested candidates to email the first author from their institutional email address. Furthermore, the first author provided each candidate with a unique alphanumeric verification code which the respondent was required to input when completing the survey. Although this decision may have impacted the quantity and demographics of respondents, we favored this procedure over adding a link to the survey in our recruitment advertisements, to ensure to the best of our ability that responses were genuine.

We asked respondents to complete three programming tasks. Analyzing the task *answers* would provide insight on whether respondents were able to successfully leverage the information presented in the tutorial to complete the tasks correctly. However, we chose not to report on the correctness of task answers, and use only the control questions and participation criteria to filter invalid responses. We made this decision because our goal was to understand *how useful* programmers found the different modalities, based on their needs and preferences. Thus, in our study, the tasks only acted as an instrument to provide a common context to all respondents while they navigated the multimodal tutorial.

We created three prototypes for three different programming topics. Although we could have created a single prototype and reported on its results, we chose to deploy multiple prototypes to account for potential bias of the programming topic to survey responses. A consequence of this decision was that there could be slight variations in difficulty between tasks in the three surveys, that may have been further compounded by respondents' prior programming experience. However, the results of the statistical tests between modality rating and topics, for each modality, indicate that there is no statistically significant association between these two dimensions except for tables. Still, we found that some respondents struggled with the inheritance debugging question (described in Section III-C). This may be because the question required some inference from the tutorial content, which could have been easy to miss.

[3] https://doi.org/10.5281/zenodo.14623560

III. Interacting with the Multimodal Tutorial

We describe our observations of the modality ratings for the different task types and topics, with insights from respondents' text responses. We refer to respondents as R#, I#, or E# according to the survey topic: Regular expressions, Inheritance, or Exception handling, respectively.

A. Modality Ratings for Conceptual Tasks

For all three topics, more respondents found text content useful compared to other modalities for the conceptual tasks (see Figure 4). The residuals reflect this observation: text content being *very useful* is observed more than expected (see Figure 5a). Respondents rationalized that text content was relevant specifically for a conceptual problem: "Because this was a more theoretical question about the usage of the "final" keyword, I was looking for information provided as an explanation" [I11].

For the conceptual tasks, respondents found regular code examples the next most useful, after text content, for regular expressions and exception handling (see Figure 4): "Text content was useful at explaining in greater detail the definition and usage of exceptions. The regular code example was very useful to get the general setup of an exception." [E10] Regular code examples being *moderately useful* are also observed more than expected, and thus have an effect on the statistical significance between modalities and their rating for the conceptual tasks (see Figure 5a).

For inheritance, the tables were the second-most useful modality (see Figure 4). The table acted as a concise source of information: "I read through the tutorial and got most of the content from the text itself. I took a look at the tables to get a summarised view of the text I just read. This was particularly useful to gather my thoughts and solidify my understanding of the material." [I10] Additionally, for some respondents: "The table had all the information I needed to answer the question, so I did not have to read on." [I17].

Observation 1: Text content was more useful than other modalities for conceptual tasks, irrespective of the topic.

B. Modality Ratings for How-to Tasks

More respondents indicated that regular code examples were useful for how-to tasks than any other modality (see Figure 4). The residual for regular code examples being *very useful* for how-to tasks indicate a larger frequency than expected by chance (see Figure 5b). Respondents explained that the how-to tasks involved programming, which made it was necessary to get an idea of a working example, which the code examples could provide: "For implementation [...] It was far more useful and relevant to see code in context [...]" [R4]

We note that many respondents indicated that they already knew the answer for the how-to tasks for inheritance and exception handling. Six respondents for inheritance and eight for exception handling relied on their prior knowledge in how-to tasks. For comparison, no respondents indicated relying on their prior knowledge for regular expressions. When recalling their knowledge, respondents only needed a reminder of the underlying concept or syntax, which the regular code example could provide: "I went straight to a code example to get a refresher on the proper syntax to be used when extending a class. The regular code example provided enough information for me, and so I didn't check the other more detailed examples." [I10]

Observation 2: Regular code examples were more useful than other modalities for how-to tasks.

C. Modality Ratings for Debugging Tasks

For debugging tasks, there is no particular modality which was most useful across all three programming topics (see Figure 4), nor are there statistically significant associations between modalities and their ratings. For regular expressions, more respondents found tables useful: "The table was definitely the most useful for this question since most of the information for the difference [between] quantifiers and metacharacters were found in the tables." [R8] However, both Figures 4 and 5c show that tables were largely ignored by respondents for inheritance and exception handling.

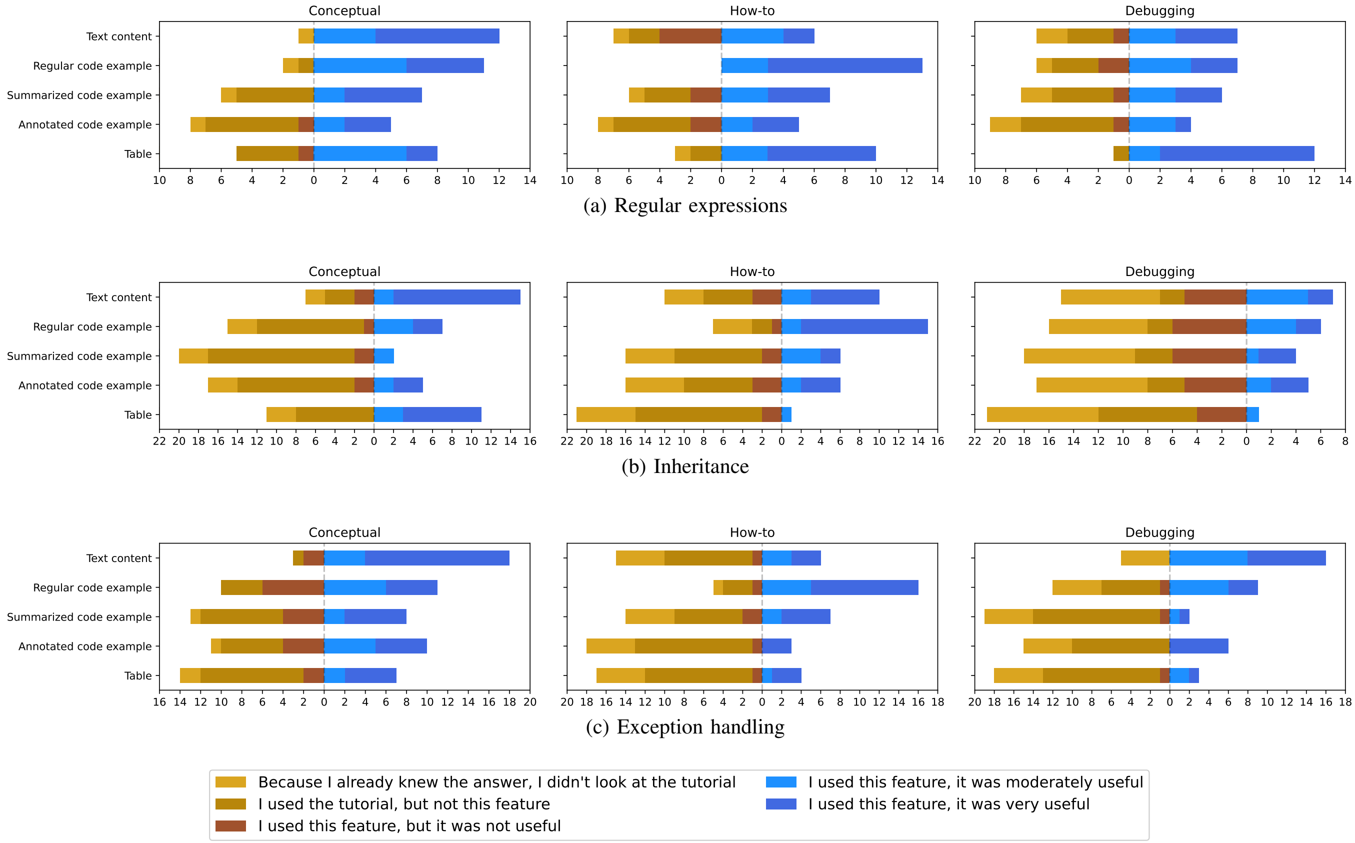
For inheritance, respondents used a combination of modalities: "It was a complex question, and initially, it wasn't clear to me why this [the issue in the task] was happening. I re-read the text to figure out what I was missing, then reviewed the code to understand how the method was being overridden, and finally, examined the table to identify the relationships between them" [I15], which explains the more balanced rating amongst the different modalities. Nine of the 22 respondents for inheritance described some difficulty with this task, either indicating they could not understand what the issue was or find the answer in the tutorial. This also explains why more respondents found the modalities *not useful* for the inheritance debugging task compared to any of the other tasks. Still, we do not observe a statistically significant association between programming topic and modality rating for all modalities except tables.

More respondents found text content useful for the exception handling debugging task, complemented by code examples. We also observed that, as for how-to tasks, more respondents already knew the answers to debugging questions than for conceptual questions (see Figure 4). Furthemore, some respondents were able to leverage their prior knowledge, and the text content provided sufficient information for them to recall and answer the question: "I was already familiar with the concept of multiple catches. I quickly checked my understanding in the textual description." [E4] Further developments on multimodal tutorials could involve incorporating additional modalities, to study whether they may be preferred for debugging tasks.

Observation 3: No modality dominated as a preference for debugging tasks, across programming topics.

D. Usefulness of Individual Modalities

Our results from Sections III-A to III-C indicate that some modalities may be favored for some task types, for some programming topics. However, we observed that at least some respondents found each modality useful. We describe the contexts in which each modality can be useful, based on how respondents explained their varying usage of the modalities in their text responses.



(a) Regular expressions

(b) Inheritance

(c) Exception handling

Fig. 4: Rating of usefulness for the five modalities, per task type and topic, for the three multimodal tutorials. Note that the legend refers to modalities as “features” (see Section II-B).

Text content was useful for understanding the underlying working and background of a technical concept: “The text helped me figure out what was going on behind the scenes of the code and to learn about the theory behind overriding methods.” [I10] Furthermore, text content complemented other modalities: “reading the small description in the table seems quicker and easier than reading the whole text. I thought if I can’t find the info in the table then I’ll read the text.” [I8] However, as the content is descriptive in nature, it does not provide the implementation know-how needed to code: “The text content provided a good theoretical background and context for the task and helped in understanding the concepts but was not as directly applicable as the code examples.” [R7]

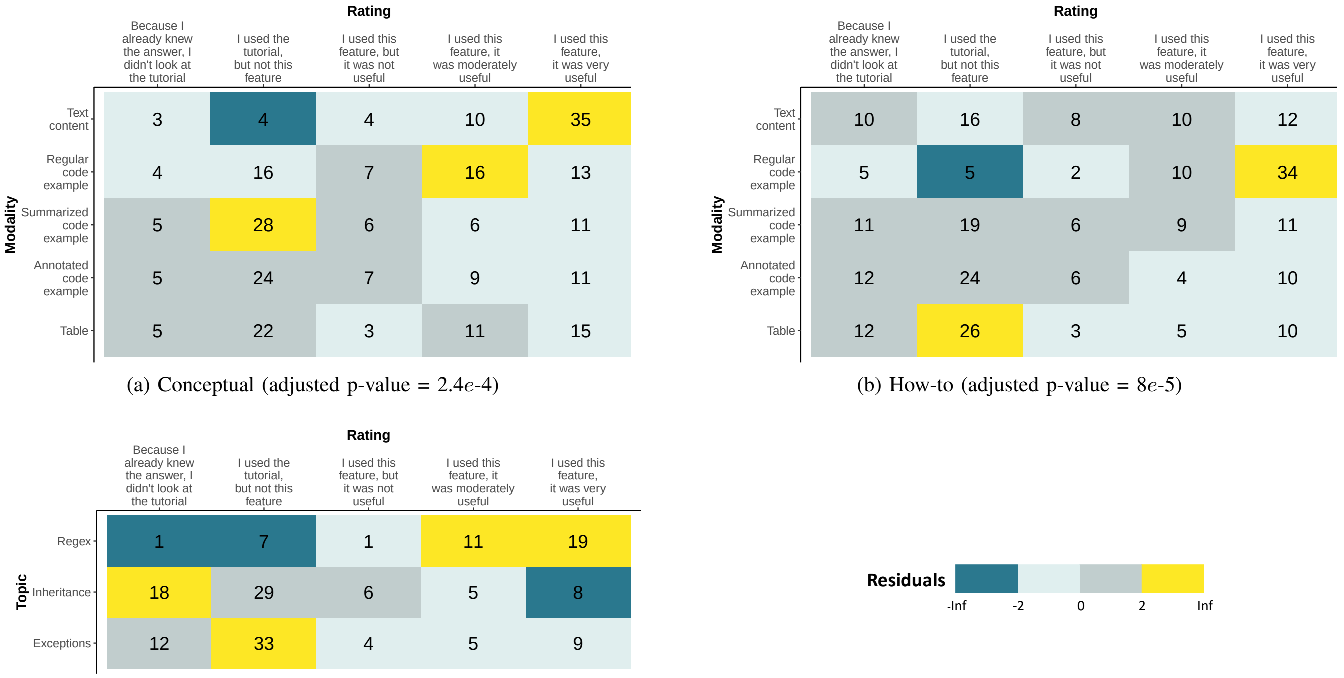
Regular code examples provided a departure point for completing coding tasks: “The code examples were useful to base my answer off of, I was able to know the syntax and number of parameters of the methods I wanted to use quickly.” [R2] Although the same code is available through annotated code examples, someone with a background in programming could find that the regular code examples were sufficient: “Annotated code was not useful for me since I understood the regular code directly [...]” [I8]

Summarized code examples “provided a concise and clear illustration of the key points and made it easier to grasp the differences without getting bogged down in too much detail.” [R7] This focus helped when programmers needed to recall information quickly: “I used the summarized code to refresh my memory on the try/catch syntax in Java. It was more concise than the regular and annotated code samples, which made it easier to find the information I was looking for.” [E11] However, others found that: “the summarized code example was missing essential code found in the regular code section.” [E5]

Annotated code examples “included comments and explanations for each part of the code and made it easier to understand the logic and purpose behind each step, enhancing the learning experience.” [R7] Although the combination of text content and regular code examples may provide sufficient information, the annotated code examples provided example-specific descriptions, which can be especially useful for beginners: “I used the annotated code example because I think it’s more readable and well explained. This feature, especially for a beginner or for someone not used to writing code in a certain language, allows the user to understand better.” [I19]

Tables were useful to have a concise overview of information present in the text content: “I first read the text, and it gave me an overview. Then I read the table, and the information was presented in a clear, concise, and more visually pleasing way.” [I6] Tables also provided a quick reference: “The table was useful for quickly understanding what each relevant method achieves.” [R4]

Although the modalities were useful for different purposes, respondents **used a combination of the modalities** to complete the programming tasks: “[The modalities] were all equally



(a) Conceptual (adjusted p-value = 2.4e-4)

(b) How-to (adjusted p-value = 8e-5)

(c) Table (adjusted p-value = 8e-5)

Fig. 5: Adjusted Standardized Residuals and contingency tables between Modality and Rating for Conceptual and HowTo programming tasks, as well as between Topic and Rating for Tables. Note that the labels refer to modalities as "features" (see Section II-B).

useful as they all provided an explanation about [the task solution] or an example which made it clear." [R10] Although containing the same information, the variations in presentation allowed the modalities to complement one another: "The text content and table allowed me to know where to look for the information I needed and the code gave a useful example." [I13] However the combinations of which modalities to use varied depending on the respondent: "I relied on text for the main of the information and then looked for practical applications of what was described in the text in summarized code blocks." [E18]; "The text content was giving useful explanations. The annotated code example gave more explanation on the example." [E16]

Observation 4: Different modalities complement one another to support comprehension from multiple perspectives, i.e. concept understanding, quick referencing, and code implementation and rationale.

E. Usefulness of Additional Tutorial Features

We report on the two optional questions about how respondents used the table of contents and the collapse and expand features, for which 55 of the 56 respondents provided an answer.

Table of contents: A total of 39 respondents found the table of contents useful to get an idea of which sections of the tutorial were relevant to the programming task, and navigate to them directly. However, this required some intuition based on where they could expect the content to be. For two respondents who were not familiar with the topic or with Java, the table of contents was not as helpful: "I skimmed everything in the tutorial because even the headings were unfamiliar to me so [the table of contents] didn't help me search because I didn't know what [the sections] were yet." [R13] Four respondents who did not use the table of contents, described that the tutorial was short and concise enough to navigate directly: "No [I did not use the table of contents], though I definitely could have if the tutorial was longer. It was short enough that I could scroll through and just read the topics that I needed." [E9] If ever needed, respondents could simply use the default webpage search functionality: "I find it easier to directly search for what I want using `ctrl+F`, since all info is on one page." [I16]

Collapse/expand: The feature to collapse and expand tutorial modalities "made the website slightly less overwhelming by collapsing things, and the expanding helped when I needed something explained." [E15] This was useful when relevant information was present across sections that were not placed next to each other: "[it] made it easier to navigate and have relevant information on the screen, independent of if it was the first and last section or the second and third section, for example." [R6] However, 39 respondents did not use the functionality because the table of contents provided sufficient navigation to allow skipping irrelevant sections: "I did not need to collapse and expand sections, tables, or code examples since the table of contents allowed me to jump directly to the sections I needed." [R7] Additionally, the tutorial was concise enough to skim through manually: "I'm sure [the collapse/expand features] are helpful for longer tutorials. This is pretty short so I did not need to do so." [E9] Seven respondents preferred not to collapse sections: "I like having everything displayed so I can be sure I am not missing anything." [I4] E14 and E18 suggested that everything be collapsed first and then a user could expand as they went: "I don't think the collapse were useful, primarily because they are already all expanded. If they started by being collapsed by default, it might have been useful, but the call to action currently is to collapse information, which is not very relevant for the user trying to access information." [E18]

Observation 5: Respondents appreciated the ability to manipulate the tutorial to gain an overview of the content or to focus on particular parts of the tutorial.

F. Recommendations from Respondents

Respondents had suggestions for how the tutorial could be improved, such as including other modalities: "More charts and diagrams/pictures would be useful. I find a combination of different "materials" helps me absorb information better [...] more images could always help" [I10]; "I like when tutorials suggest a little project/example for you to try out." [I17], and an in-page integrated code editor and runner (R10, R13, I14, I24, E9, E13). These suggestions motivate the need for multiple modalities in tutorials: "I liked the different ways that the information was conveyed! I like having options to best fit my specific needs." [I6]

Contradictions in preferences between participants surface the need for adaptable tutorials whose design can be customized by users. For example, some respondents appreciated the annotated code examples (I10, I16, I23, E13, E16), even wondering: "Not sure why keep Regular [code example] as default when Annotated is superior." [E13] However, one respondent would do away with them entirely: "I wish [the tutorial] was just headers (to navigate), text (to understand theoretical concept) and summarized code blocks (to understand practically in code)." [E18] Another respondent wanted the ability to collapse all the code at once: "I think a button to collapse all the code at once would make navigation easier since they take up a lot of space." [R12] Respondents also wanted variations in other user experience aspects: "I would maybe just change the colours since I don't like websites that are beige (maybe like a dark and light mode) but that's only a personal preference." [I8]

Observation 6: Respondents had contradictions in their preferences, motivating the need for customizable documentation.

IV. Related Work

We discuss our results in the context of prior work on *information needs and preferences* and *tools to support user control of navigating documentation*.

A. Information Needs and Preferences

Users' search for information begins with an *information need* that must be fulfilled. There are multiple types of questions that developers need answered during everyday development and maintenance activities, largely related to code behaviour [11], [15], [52]. In the context of every-day information search, prior work has explored the information needs of developers based on their web searches [18], [27].

In addition to information needs, programmers may have *preferences* about the resources they refer to. Escobar-Avila et al. surveyed 205 Computer Science (CS) students and professionals to determine their habits in learning programming and its related concepts [16]. More than 55% in both populations said they preferred visual and auditory formats for learning, and only about 3% indicated they preferred text-only mediums. Particularly, when learning a programming-specific or CS-related concept, most respondents used tutorials and code examples, irrespective of the target programming language. Our study corroborates the results of Escobar et al.: for conceptual tasks, respondents favored using text content, and used code examples to strengthen their understanding of the concepts. Our findings also indicate that a preference for code examples exists for how-to tasks, whereas for debugging tasks, there is no preferred information modality. Additionally, respondents leveraged modalities that can complement their own prior knowledge. For example, if they had an understanding of a concept, then they referred to only code examples to refresh their knowledge of programming syntax.

From a survey of 74 individuals at an IBM enterprise customers event, Earle et al. reported that 59 of the 64 responses to the survey question "*How important to you is the format of the information?*" indicated 3 and above on a five-point increasing scale of importance [13]. They found that tech notes and videos were the most preferred formats among these respondents. Furthermore, respondents' preferences for formats in software product documentation differed based on their role and responsibilities. For example, administrators who maintain multi-user systems refer to a wider range of documentation elements, such as product help systems, tech notes, and forums, in comparison to architects who focus on design, and refer primarily to articles. The diversity of tasks [32] that the role of "software engineer" involves [35], and the variations of modality use based on the programming task types (see Section III), indicates the need to have documentation that can be organized in a flexible manner.

To cater to varying needs, documentation creators are forced to manage multiple formats of software documentation [12] to avoid information inconsistency [4], [5]. With feedback from users on their needs and preferences, they may even rewrite user manuals and reorganizing the content, which are time and resource-intensive activities [42]. Thus, software documentation creation can be a tiresome process [3]. Our results indicate that programmers' preferences may be contradictory, making it a further challenge to create documentation that can cater to all audience needs. Instead, our findings point towards the need for multimodal tutorials that contain all relevant information in different presentation formats, allowing users to gather the information they need, in the way that they prefer, without additional strain on documentation creators.

B. User Controls for Navigating Documentation

To find relevant information, developers use different strategies, leveraging their knowledge about where to look for information [31]. Software developers may also use *cues* such as *creation time*, when searching among multiple similar code snippets [46], or judge the potential value of text based on its style [34]. Prior work has studied how to support information seeking with explicit cues, such as indicating the time cost of reading a resource [24], providing an overview of blog comments [20] or of all pages in a document [19].

Allowing users to use categories to filter the information that they need through buttons is known as *faceted browsing*. This browsing technique has been proposed as a means to support users in finding the information they need effectively [9], [14], [29], [36], [57]. Käki and Aula [25], and Käki [26] performed user studies in a laboratory and natural setting to evaluate their tool Findex, which categorized search results that users could use to filter their searches.

Liu and Holmes investigated two information representations in integrated development environments (IDEs): (1) *inline views* where information is presented anchored to the source code, and (2) *isolated views* where information is presented in a separate area, such as a notification panel [33]. The authors conducted a survey with four tasks to determine developers' preferences for either view. Whereas some participants appreciated the minimalistic nature of the inline view, others preferred having access to additional information via a separate panel because they could choose to look at it when they needed to. In our study, too, we report contradictory modality preferences between respondents, that can prove challenging for documentation design.

Adenuga et al. proposed a "Living document" system that generates text summaries from existing online articles based on an input topic prompt, and allows user to manipulate these summaries, for example by inserting and removing sentences [2]. The authors evaluated the system via a user study wherein 25 participants were asked to create summaries about a pre-defined topic related to either science or sports, using the Living Document summarizer, and share their insights on using the system. Nine participants indicated that the system responded to their interactions adequately, thus giving them a sense of control. However, other participants described that changes in the text were subtle, giving the impression that their interactions did not make a difference on the text. The work by Adenuga et al. shows promise for user-controlled documents, but highlights the challenge of providing user control, i.e. making a document clearly manipulatable at the smallest units of change possible. Personalization of a document would need to involve the construction of a user profile, which could pose a threat to user privacy [50].

Our observations, supported by the findings from prior literature, indicate the need to shift the control of content visibility in documentation customization towards the user. Such control can allow a multimodal documentation to reproduce typical documentation styles as they are known today. For example, for quick referencing, a programmer can collapse all other modalities except tables to emulate the typical presentation style of application programming interface documentation. This can be applied immediately, such as changes to settings of color shown immediately to users [30]. Alternatively, configuration options can be selected and accumulated, and then applied all together allowing a programmer to directly see a new documentation build, at once [45].

V. Conclusion

Given that users may have different information needs and that there are multiple ways to present information, we studied how programmers make decisions about their presentation needs and preferences when accessing software documentation. We developed three multimodal tutorials on three programming concepts in Java, namely regular expressions, inheritance, and exception handling. In each tutorial, we provided five modalities, i.e., text content, regular code examples, summarized code examples, annotated code examples, and tables. Through a survey study, we asked programmers to use one of the multimodal tutorials and complete three programming tasks, one of each type: conceptual, how-to, and debugging. We observed that respondents preferred text content for conceptual tasks and regular code examples for how-to tasks, with no clear modality preference for debugging tasks. Still, the variations in responses indicate that there are no universal modality preferences for all software programming contexts. Our results corroborate our hypothesis that having multiple modalities within a single document can serve diverse information needs for programming tasks. Further research could help assess the potential of multimodal documentation for other programming languages and software technologies.

Our proposed multimodal tutorial is an initial step towards the user interface of multimodal, multifeatured software documentation. A multimodal design allows resource creators to create a single, complete, consistent, and thorough document. Then, information seekers, aware of their needs and preferences specific to the task at hand, can configure the documentation to present the information in pertinent modalities and with the relevant visibility. Future work could investigate the ability to save multimodal preferences for further use, such that documentation can be conveniently personalized. Further research could also explore the ability to apply multimodal documentation in other developer workflows, such as in integrated development environments (IDEs) and in developer tools such as documentation generators.

VI. Acknowledgements

We thank Vivian Li for their assistance in building the foundation for the multifaceted tutorial prototypes, and Shushi Huang for the further development of the prototypes and input in creating the programming tasks for the survey. We also thank the reviewers for their insightful feedback. This work is funded by the Natural Sciences and Engineering Research Council of Canada (NSERC).

REFERENCES

[1] Hervé Abdi. Bonferroni and Šidák corrections for multiple comparisons. *Encyclopedia of measurement and statistics*, 3:103–107, 2007.

[2] Iyadunni J. Adenuga, Benjamin V. Hanrahan, Chen Wu, and Prasenjit Mitra. Living documents: Designing for user agency over automated text summarization. In *Proceedings of the Extended Abstracts of the Conference on Human Factors in Computing Systems*, pages 1–6, 2022.

[3] Emad Aghajani, Csaba Nagy, Mario Linares-Vásquez, Laura Moreno, Gabriele Bavota, Michele Lanza, and David C. Shepherd. Software documentation: The practitioners' perspective. In *Proceedings of the International Conference on Software Engineering*, pages 590–601, 2020.

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

[5] Deeksha M. Arya, Jin L. C. Guo, and Martin P. Robillard. Information correspondence between types of documentation for APIs. *Empirical Software Engineering*, 25:4069–4096, 2020.

[6] Deeksha M. Arya, Jin L. C. Guo, and Martin P. Robillard. How programmers find online learning resources. *Empirical Software Engineering*, 28(3), 2022.

[7] Deeksha M. Arya, Jin L. C. Guo, and Martin P. Robillard. Properties and styles of software technology tutorials. *IEEE Transactions on Software Engineering*, 50(2), 2024.

[8] Deeksha M. Arya, Mathieu Nassif, and Martin P. Robillard. A data-centric study of software tutorial design. *IEEE Software*, 39(3), 2022.

[9] Sven Buschbeck, Anthony Jameson, Adrian Spirescu, Tanja Schneeberger, Raphaël Troncy, Houda Khrouf, Osma Suominen, and Eero Hyvönen. Parallel faceted browsing. In *Proceedings of the Extended Abstracts on Human Factors in Computing Systems*, pages 3023–3026, 2013.

[10] Shi-Yi Chen, Zhe Feng, and Xiaolian Yi. A general introduction to adjustment for multiple comparisons. *Journal of Thoracic Disease*, 9(6), 2017.

[11] Ekwa Duala-Ekoko and Martin P. Robillard. Asking and answering questions about unfamiliar APIs: An exploratory study. In *Proceedings of the International Conference on Software Engineering*, pages 266–276, 2012.

[12] Koznov D.V., Luciv D.V., and Chernishev G.A. Duplicate management in software documentation maintenance. In *Proceedings of the International Conference on Actual Problems of System and Software Engineering*, pages 195–201, 2017.

[13] Ralph H. Earle, Mark A. Rosso, and Kathryn E. Alexander. User preferences of software documentation genres. In *Proceedings of the Annual International Conference on the Design of Communication*, pages 1–10, 2015.

[14] Jennifer English, Marti Hearst, Rashmi Sinha, Kirsten Swearingen, and Ka-Ping Yee. Hierarchical faceted metadata in site search interfaces. In *Proceedings of the Extended Abstracts on Human Factors in Computing Systems*, pages 628–639, 2002.

[15] K. Erdos and H. M. Sneed. Partial comprehension of complex programs (enough to perform maintenance). In *Proceedings of the International Workshop on Program Comprehension*, pages 98–105, 1998.

[16] Javier Escobar-Avila, Deborah Venuti, Massimiliano Di Penta, and Sonia Haiduc. A survey on online learning preferences for computer science and programming. In *Proceedings of the International Conference on Software Engineering: Software Engineering Education and Training*, pages 170–181, 2019.

[17] Andrew Forward and Timothy C Lethbridge. The relevance of software documentation, tools and technologies. In *Proceedings of the ACM Symposium on Document Engineering*, pages 26–33, 2002.

[18] Rosalva E. Gallardo-Valencia and Susan Elliott Sim. What kinds of development problems can be solved by searching the web?: A field study. In *Proceedings of the International Conference on Software Engineering*, pages 41–44, 2011.

[19] Carl Gutwin, Andy Cockburn, and Nickolas Gough. A field experiment of spatially-stable overviews for document navigation. In *Proceedings of the Conference on Human Factors in Computing Systems*, pages 5905–5916, 2017.

[20] E. Hoque and G. Carenini. Convis: A visual text analytic system for exploring blog conversations. *Computer Graphics Forum*, 33(3):221–230, 2014.

[21] Andre Hora. Googling for software development: What developers search for and what they find. In *Proceedings of the IEEE/ACM International Conference on Mining Software Repositories*, pages 317–328, 2021.

[22] Matthew Hurst. Towards a theory of tables. *International Journal of Document Analysis and Recognition*, 8:123–131, 2006.

[23] Mohieddin Jafari and Naser Ansari-Pour. Why, when and how to adjust your p values? *Cell Journal (Yakhteh)*, 20(4), 2018.

[24] Xiaoyu Jin, Nan Niu, and Michael Wagner. Facilitating end-user developers by estimating time cost of foraging a webpage. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing*, pages 31–35, 2017.

[25] Mika Käki. Findex: Search result categories help users when document ranking fails. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 131–140, 2005.

[26] Mika Käki and Anne Aula. Findex: Improving search result use through automatic filtering categories. *Interacting with Computers*, 17(2):187–206, 2005.

[27] Amy J. Ko, Robert DeLine, and Gina Venolia. Information needs in collocated software development teams. In *Proceedings of the International Conference on Software Engineering*, pages 344–353, 2007.

[28] Amy J. Ko, Brad A. Myers, Michael J. Coblenz, and Htet Htet Aung. An exploratory study of how developers seek, relate, and collect relevant information during software maintenance tasks. *IEEE Transactions on Software Engineering*, 32(12):971–987, 2006.

[29] Bill Kules and Ben Shneiderman. Users can change their web search tactics: Design guidelines for categorized overviews. *Information Processing and Management*, 44(2):463–484, 2008.

[30] Gerhard Leitner, Alexander Felfernig, Paul Blazek, Florian Reinfrank, and Gerald Ninaus. *Chapter 8 - User Interfaces for Configuration Environments*, page 89–106. Morgan Kaufmann, Boston, 2014.

[31] Hongwei Li, Zhenchang Xing, Xin Peng, and Wenyun Zhao. What help do developers seek, when and how? In *Proceedings of the Working Conference on Reverse Engineering*, pages 142–151, 2013.

[32] Sherlock A. Licorish and Stephen G. MacDonell. Exploring software developers' work practices: Task differences, participation, engagement, and speed of task resolution. *Information & Management*, 54(3):364–382, 2017.

[33] Xinhong Liu and Reid Holmes. Exploring developer preferences for visualizing external information within source code editors. In *Proceedings of the Working Conference on Software Visualization*, pages 27–37, 2020.

[34] Arthur Marques, Nick C. Bradley, and Gail C. Murphy. Characterizing task-relevant information in natural language software artifacts. In *Proceedings of the IEEE International Conference on Software Maintenance and Evolution*, pages 476–487, 2020.

[35] Edward Meade, Emma O'Keeffe, Niall Lyons, Dean Lynch, Murat Yilmaz, Ulas Gulec, Rory V. O'Connor, and Paul M. Clarke. The changing role of the software engineer. In *Systems, Software and Services Process Improvement*, pages 682–694, 2019.

[36] Yevgeniy Medynskiy, Mira Dontcheva, and Steven M. Drucker. Exploring websites through contextual facets. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 2013–2022, 2009.

[37] Cyrus R Mehta and Nitin R Patel. IBM SPSS Exact Tests. *Armonk, NY: IBM Corporation*, 2011.

[38] Michael Meng, Stephanie Steinhardt, and Andreas Schubert. How developers use API documentation: An observation study. *Communication Design Quarterly*, 7(2):40–49, 2018.

[39] Aliaksei Miniukovich, Antonella De Angeli, Simone Sulpizio, and Paola Venuti. Design guidelines for web readability. In *Proceedings of the Conference on Designing Interactive Systems*, pages 285–296, 2017.

[40] Mathieu Nassif, Zara Horlacher, and Martin P. Robillard. Casdoc: unobtrusive explanations in code examples. In *Proceedings of the International Conference on Program Comprehension*, pages 631–635, 2022.

[41] Mathieu Nassif and Martin P. Robillard. A field study of developer documentation format. In *Proceedings of the Extended Abstracts of the Conference on Human Factors in Computing Systems*, pages 1–7, 2023.

[42] Aleksandra Pawlik, Judith Segal, and Marian Petre. Documentation practices in scientific software development. In *Proceedings of the International Workshop on Co-operative and Human Aspects of Software Engineering*, pages 113–119, 2012.

[43] Michael Priestley. Dynamically assembled documentation. In *Proceedings of the Annual International Conference on Computer Documentation*, pages 53–57, 1999.

[44] Daniele Procida. Diátaxis documentation framework, 2023.

[45] Rick Rabiser, Michael Vierhauser, Martin Lehofer, Paul Grünbacher, and Tomi Männistö. *Configuring and Generating Technical Documents*, pages 241–250. 2014.

[46] Sruti Srinivasa Ragavan, Sandeep Kaur Kuttal, Charles Hill, Anita Sarma, David Piorkowski, and Margaret Burnett. Foraging among an overabundance of similar variants. In *Proceedings of the Conference on Human Factors in Computing Systems*, pages 3509–3521, 2016.

[47] Nikitha Rao, Chetan Bansal, Thomas Zimmermann, Ahmed Hassan Awadallah, and Nachiappan Nagappan. Analyzing web search behavior for software engineering tasks. In *Proceedings of the IEEE International Conference on Big Data*, pages 768–777, 2020.

[48] Martin P. Robillard, Deeksha M. Arya, Neil A. Ernst, Jin L.C. Guo, Maxime Lamothe, Mathieu Nassif, Nicole Novielli, Alexander Serebrenik, Igor Steinmacher, and Klaas-Jan Stol. Communicating study design trade-offs in software engineering. *Transactions on Software Engineering and Methodology*, 33(5), 2024.

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

[50] Martin P. Robillard, Andrian Marcus, Christoph Treude, Gabriele Bavota, Oscar Chaparro, Neil Ernst, Marco Aurélio Gerosall, Michael Godfrey, Michele Lanza, Mario Linares-Vásquez, Gail C. Murphy, Laura Moreno, David Shepherd, and Edmund Wong. On-demand developer documentation. *Proceedings of the International Conference on Software Maintenance and Evolution*, pages 479–483, 2017.

[51] Donald Sharpe. Chi-square test is statistically significant: Now what? *Practical Assessment, Research, and Evaluation*, 20(1):8, 2015.

[52] Jonathan Sillito, Gail C. Murphy, and Kris De Volder. Questions programmers ask during software evolution tasks. In *Proceedings of the SIGSOFT International Symposium on Foundations of Software Engineering*, pages 23–34, 2006.

[53] Rebecca Tiarks and Walid Maalej. How does a typical tutorial for mobile development look like? In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 272–281, 2014.

[54] Robert B. Watson. Development and application of a heuristic to assess trends in API documentation. In *Proceedings of the ACM International Conference on Design of Communication*, pages 295–302, 2012.

[55] S. Paul Wright. Adjusted p-values for simultaneous inference. *Biometrics*, 48(4):1005–1013, 1992.

[56] Xin Xia, Lingfeng Bao, David Lo, Pavneet Singh Kochhar, Ahmed E. Hassan, and Zhenchang Xing. What do developers search for on the web? *Empirical Software Engineering*, 22(6):3149–3185, 2017.

[57] Ka Ping Yee, Kirsten Swearingen, Kevin Li, and Marti Hearst. Faceted metadata for image search and browsing. In *Proceedings of the Conference on Human Factors in Computing Systems*, pages 401–408, 2003.

[58] Annie T. T. Ying and Martin P. Robillard. Selection and presentation practices for code example summarization. In *Proceedings of the International Symposium on Foundations of Software Engineering*, pages 460–471, 2014.