

EMPIRICAL FOUNDATIONS FOR SOFTWARE DOCUMENTATION DESIGN

Deeksha Arya

School of Computer Science
McGill University, Montreal, Canada

July 23, 2025

A thesis submitted to McGill University in partial fulfillment of
the requirements of the degree of Doctor of Philosophy

© Deeksha Arya, 2025

Contents

Contents	i
Abstract	v
Résumé	vii
Contributions	ix
Acknowledgements	xii
List of Figures	xiv
List of Tables	xvi
List of Acronyms	xviii
1 Introduction	1
1.1 Thesis Organization	9
2 Background and Related Work	10
2.1 Documentation Seeking	11
2.1.1 Information Needs	11
2.1.2 Resource Design Preferences	12
2.1.3 Online Resource Seeking	13
2.1.4 Identifying Pertinent Information	13
2.2 Documentation Contribution	15
2.2.1 Documentation Motivations	15
2.2.2 Documentation Creation Practices	16
2.2.3 Mindsets in Software Engineering	16
2.3 Design of Documentation	17
2.3.1 Characteristics of Documentation	18
2.3.2 Formal Concept Analysis	19
2.3.3 User Controls for Navigating Documentation	19

3	How Programmers Find Software Documentation	21
3.1	Study Design	22
3.1.1	Data Collection	22
3.1.2	Data Analysis	24
3.2	Resource-Seeking Model	27
3.2.1	Need-oriented Components	27
3.2.2	Resource-oriented Components	29
3.3	Need-Oriented Components	30
3.3.1	QUESTIONS	30
3.3.2	PREFERENCES	31
3.3.3	BELIEFS	32
3.4	Resource-Oriented Components	32
3.4.1	RESOURCES	32
3.4.2	CUES	34
3.4.3	IMPRESSION FACTORS	36
3.5	Relations Between Components	37
3.5.1	RESOURCE <i>is accessed for</i> QUESTION	39
3.5.2	CUE <i>is used to select</i> RESOURCE	41
3.5.3	RESOURCE <i>is evaluated through</i> IMPRESSION FACTOR	42
3.5.4	Infrequent Relations	42
3.6	Implications	45
4	Documentation Properties and Styles	48
4.1	Data Collection	49
4.1.1	Resource Collection	49
4.1.2	Property Extraction	50
4.1.3	Limitations	52
4.2	Resource Properties	54
4.2.1	Variations in Property Values	55
4.2.2	Correlations Between Properties	58
4.2.3	Correspondence of Properties to Website Traffic	59
4.2.4	From Properties to Styles	59
4.3	Characterizing Resources	59
4.3.1	Prominent Style	61
4.3.2	Recurring Style	62
4.3.3	User-defined Style	65
4.3.4	Discussion	66
4.3.5	Limitations	66
5	Considerations of Documentation Creators	68
5.1	Study Design	69
5.1.1	Informant Recruitment	69
5.1.2	Data Collection	70

5.1.3	Qualitative Analysis	70
5.1.4	Mindset Elicitation	71
5.1.5	Validation	72
5.2	Dimensions of the Software Documentation Contribution Process	73
5.2.1	Motivations	73
5.2.2	Topic Selection Techniques	75
5.2.3	Styling Objectives	77
5.3	Software Documentor Mindsets	80
5.4	Validation	85
5.4.1	Study Design Trade-offs	87
5.5	Implications	87
5.5.1	Balancing Multiple Mindsets	87
5.5.2	Challenges with Pursuing Considerations	88
5.5.3	Other Mindsets	89
6	Interactions with Multimodal Documentation	91
6.1	Study Design	92
6.1.1	Multimodal Tutorial Prototype	92
6.1.2	Survey Design	94
6.1.3	Respondent Recruitment	96
6.1.4	Analysis	97
6.1.5	Study Design Trade-offs	98
6.2	Programmer Interactions with the Multimodal Tutorial	99
6.2.1	Modality Ratings for Conceptual Tasks	99
6.2.2	Modality Ratings for How-to Tasks	102
6.2.3	Modality Ratings for Debugging Tasks	102
6.2.4	Usefulness of Individual Modalities	103
6.2.5	Usefulness of Additional Tutorial Features	104
6.2.6	Recommendations from Respondents	105
7	Discussion	106
7.1	The Software Documentation Environment	106
7.1.1	Software Documentation is Human-centric	107
7.1.2	Management of Multiple Documentation Types	108
7.1.3	Design of Customizable Documentation	109
7.2	Anticipated Challenges to Designing Customizable Documentation	110
7.2.1	Shift of Design Effort from Documentor to Programmer	110
7.2.2	Evolving Design Needs and Preferences	111
7.2.3	Evaluation of Software Documentation Quality	112
7.2.4	Impact of Artificial Intelligence on Documentation Creation and Use	113
7.3	Future Research Directions	114
7.3.1	Code Example Customization	114
7.3.2	Querying Pertinent Documentation	115

7.3.3	Other Software Documentation Design Considerations	117
7.3.4	Communication Between Documentation Creators and Information Seekers	118
8	Conclusion	119
	Bibliography	123
A	Replication Package for How Programmers Find Online Learning Re- sources	150
A.1	Demographic Questions	151
A.2	Post-study Questionnaire	151
B	Replication Package for Properties and Styles of Software Tutorials	152
B.1	Resource Collection	154
B.2	Recurring Resource Styles	154
C	Replication Package for The Documentor Mindset	156
C.1	Interview Guide	156
C.2	Validation Questionnaire	157
D	Replication Package for How Programmers Interact with Multimodal Doc- umentation	161
D.1	Tasks per Topic	161
D.2	Fisher's Test Results	164
D.3	Contingency Tables	165

Abstract

Software documentation provides information about the details of a software technology and how it can be used. Documentation creators must invest time and effort to make decisions about the content, organization, and presentation of the information. The outcomes of different decisions lead to a wide variety in the resources available. Although search engines are helpful to filter relevant resources, programmers are still required to manually browse through them before they find the information most pertinent to their needs. Prior work has focused on helping programmers find the location of information within a resource. Recent research has also begun exploring the ability to summarize information related to a programmer’s search query. These approaches support the search for information content, and thus rely on matching the search query. However, the techniques do not necessarily account for other aspects of a programmer’s search context, such as their preferred documentation design.

In this thesis, we focus on understanding how software documentation can be designed to support programmers’ search for pertinent information, while considering documentation creators’ concerns. This research took place in four phases. In the first phase, we investigated how programmers find software documentation resources online. We conducted a diary study in which participants self-reported the steps they took while searching online and their rationale behind these steps, when learning a new technology. We analysed the entries and described how programmers used “cues” to locate the resource that would best suit their need. We contribute a theoretical model that describes the different components of the resource-seeking process, and provides insight on what programmers consider when searching for information.

In the second phase, we studied the variations in the current software documentation landscape. We analysed how software tutorials vary in their properties, such as the amount of text to code ratio they contain. We reported on how tutorials can be distinguished based on their properties, and identified as being of a particular style. We contribute an overview of the design of software tutorials and a systematic method to characterize tutorials. Our framework provides a formal technique to define a software tutorial based on its design.

The third phase involved understanding why and how people voluntarily contribute documentation online, via interviews with 26 documentors. We identified sixteen considerations that the documentors had, which impacted the decisions they made, including how they selected topics and their objectives when styling their documentation. We contribute a framework of five mindsets that documentors have during documentation contribution, that

are based on related considerations. Our findings provide insight on what the documentation contribution entails, and how documentors can be supported in tailoring their documentation to intended audiences.

In the final phase, we leveraged our findings from the previous three phases: we contribute a prototype multimodal documentation that allows information seekers to select information from different presentation formats. Multimodality in documentation would allow documentors to create a single all-encompassing document, avoiding the need to manage multiple documentation types to reach varied audiences. We studied how programmers would interact with the documentation, and reported on the usefulness of the modalities for three types of programming tasks.

Our results from the four phases of the thesis show promise for versatile, customizable documentation. Such documentation provides control to users to manipulate it to their needs, while allowing documentation creators to focus on curating important information.

Résumé

Documentation de logiciel fournit des informations sur les détails de la technologie logicielle. Les auteurs de documentation doivent investir des efforts pour prendre plusieurs de décisions concernant le contenu, l'organisation et la présentation de l'information. Les différences entre les décisions prises mènent à une grande variété de ressources documentaires disponibles. Bien que les moteurs de recherche soient utiles pour filtrer les ressources pertinentes, les programmeurs doivent néanmoins parcourir différentes ressources manuellement avant de trouver les informations les plus intéressantes pour leurs besoins et leur contexte de programmation. La recherche antérieure s'est largement consacré à aider les programmeurs à localiser précisément les informations pertinentes au sein d'une ressource. Des travaux récentes ont également exploré la possibilité de résumer les informations liées à la requête de recherche d'un programmeur. Cependant, les techniques ne tiennent pas toujours compte d'autres aspects du contexte de recherche d'un programmeur, comme la conception de documentation préféré.

Dans cette thèse, nous nous concentrons sur la manière dont la documentation logicielle peut être conçue pour aider les programmeurs, tout en tenant compte des contraintes des auteurs de documentation. Cette recherche s'est déroulée en quatre phases. Dans la première phase, nous avons étudié comment les programmeurs trouvent des ressources en ligne. Nous avons mené une étude qualitative dans laquelle les participants ont déclaré eux-mêmes les étapes qu'ils ont suivies lors de leurs recherches et la justification de ces étapes lors de l'apprentissage d'une nouvelle technologie. Nous avons analysé les entrées de journal et avons décrit comment les programmeurs utilisaient les "cues" pour localiser la ressource qui répondrait le mieux à leurs besoins. Nous proposons un modèle qui donne un aperçu de ce que les programmeurs considèrent lors de la recherche d'informations.

Dans la deuxième phase, nous avons étudié comment les propriétés de documentation de logiciel varient. Nous avons expliqué comment les tutoriels peuvent être discernés selon leurs propriétés et identifiés comme étant à un style particulier. Nous fournissons un aperçu de la conception de tutoriels de logiciels et une méthode systématique pour caractériser les tutoriels. Notre modèle fournit une technique formelle pour définir un didacticiel logiciel basé sur sa conception.

La troisième phase consistait à comprendre pourquoi et comment les gens contribuent volontairement à la documentation, via des entrevues avec 26 documenteurs. Nous avons identifié cinq états d'esprit des documenteurs, qui ont eu un impact sur leurs décisions, y compris la manière dont ils ont sélectionné les sujets et leurs objectifs lors de la création

de documentation. Nous décrivons un modèle conceptuel incluant les cinq états d'esprit et leurs considérations. Nos résultats donnent un aperçu de ce qu'implique la contribution à la documentation et de la manière dont les documenteurs peuvent être soutenus quand ils adaptent documentation aux publics visés.

Dans la phase finale, nous avons développé un prototype de documentation multimodale qui permet aux programmeurs de sélectionner des informations parmi différents formats de présentation. Une documentation multimodale permettrait aux documenteurs de créer un seul document global, évitant ainsi d'avoir à gérer plusieurs types de documentation. Nous avons étudié comment les programmeurs ont interagi avec la documentation et avons rendu compte de l'utilité des modalités pour trois types de tâches.

Nos résultats des quatre phases de la thèse sont encourageant le développement de documentation personnalisable. Cette documentation permet aux utilisateurs de contrôler ses propriétés selon leurs besoins, tout en permettant aux auteurs de documentation de se concentrer sur la création et la synthèse d'informations importantes.

Contributions

The research presented in this thesis makes contributions to the domain of information seeking in the context of documentation for software technologies. The thesis is a culmination of four research projects, described in Chapters 3, 4, 5, and 6.

Chapter 3 discusses our investigation of how programmers search for online documentation via a user diary study. We analysed the diary entries of ten participants to identify the steps they took and their underlying thought process in navigating among multiple search results. Our contributions from this study are:

- a theoretical model comprising of six components of how programmers seek online resources when learning a new technology, and the nine relationships between them;
- detailed insights about the resource seeking process based on our observations;
- a publicly available replication package containing the documents to conduct the diary study as well as a data set of the qualitatively analysed diary entries [17].

This study and its contributions have been published in the article *How Programmers Find Online Learning Resources* [16]. The author of this thesis was the principal investigator of the study. She designed the user study with the guidance of her supervisors. She recruited the participants and conducted the diary study. She iteratively qualitatively analysed the diary entries, based on continuous discussions with her supervisors. She wrote the original drafts of the associated paper, and edited subsequent drafts based on regular reviews from her supervisors.

Chapter 4 discusses our investigation of how software tutorials vary in their structure-based properties. This study also involved exploring how software tutorials can be characterized as being of a particular style. We analysed software tutorials from 22 websites across three programming languages. Our contributions from this study are:

- an overview of property variations across tutorials per programming language;
- a discussion of the associations between the tutorial properties per programming language;
- a discussion of associations between the tutorial properties and traffic metrics of the source website;

- a framework for characterizing resources based on their *style*, i.e. the combination of their deviating properties;
- a discussion of three context-relevant resource styles, including their motivation, applicability, and identification techniques;
- a publicly available replication package containing the data set of the tutorials and their properties and the programming scripts to run the analysis [18].

This study and its contributions have been published in the article *Properties and Styles of Software Technology Tutorials* [19]. The author of this thesis was the principal investigator of the study. She collected the tutorials through a semi-automated process and implemented the scripts to extract and present the tutorial properties. She also implemented the code to perform formal concept analysis to characterize resources based on their styles, with advice from her supervisor. She also consulted her colleague Jessie Galasso-Carbonnel who provided valuable advice about the use and interpretation of formal concept analysis. She wrote the original drafts of the associated paper, and edited subsequent drafts based on regular reviews from her supervisors.

Chapter 5 discusses our investigation of why and how people voluntarily create and contribute software documentation online. The study involved interviewing documentation contributors about their motivations and processes in contributing documentation. Our contributions from this study are:

- a detailed discussion of 16 considerations of the documentation contribution process across three dimensions;
- a framework depicting the five documentation contributor mindsets and their associated considerations across the three dimensions;
- a discussion based on the validation of the mindsets with interviewees, including additional mindsets described by interviewees;
- a publicly available replication package containing the documents to conduct the interview study as well as the data set of the results of the qualitative analysis [20].

The study of why people voluntarily contribute software documentation online was published in the article *Why People Contribute Software Documentation* [22]. The extended work including the investigation on what the documentation process entails is available as an ArXiv preprint titled *The Software Documentor Mindset* [21]. The author of this thesis was the principal investigator of the study. She developed the interview guide with guidance from her supervisors. She recruited the participants and conducted the interviews. She iteratively qualitatively analysed the interview transcripts, based on continuous discussions with her supervisors. She created the documents for validation of the mindset results with advice from her supervisors. She communicated with participants to receive validation responses, and analysed these responses. She wrote the original drafts of the associated paper, and edited subsequent drafts based on regular reviews from her supervisors.

Chapter 6 discusses our investigation of how programmers interact with multimodal software documentation, based on different contexts such as their own presentation preferences and the programming tasks they must complete. The study involved building a prototype multimodal software tutorial, and conducting a survey in which respondents used the tutorial to complete three types of programming tasks. Our contributions from this study are:

- a prototype design for a multimodal software tutorial, with three implemented examples;
- quantitative results regarding how useful different modalities are for different types of programming tasks;
- a discussion about the context in which each modality is useful;
- a discussion of how multimodal documentation can be enriched with features to support user interaction.

The study of multimodal documentation was published in the article *How Programmers Interact with Multimodal Software Documentation* [23]. The author of this thesis was the principal investigator of the study. She developed the design of the prototype tutorial. She guided the development of the prototype, which was assisted by undergraduate research interns who were supervised by her supervisors. She created the survey and its questions with guidance from her supervisors. She conducted the pilot studies, recruited the respondents, and performed the analysis of survey responses, based on continuous discussions with her supervisors. She wrote the original drafts of the associated paper, and edited subsequent drafts based on regular reviews from her supervisors.

Acknowledgements

It is quite the feat to complete a PhD, and while I have done my share of work, it is a far more collaborative experience than I first imagined. I have many people to thank for my journey over the past few years.

First and foremost, I would not be in a position to write these acknowledgements without my supervisors, Jin and Martin. I am proud to say that I am Jin's first Master's student, and also first PhD student. She has been more than just a supervisor throughout my graduate studies; she has been an unwavering source of encouragement, through all the ups and downs of my student and personal life. Her passion is infectious, and her ability to approach a problem from multiple perspectives is inspiring. Working with Jin in the first year of my graduate studies is truly what made me appreciate and enjoy research, so much so as to pursue it as a career. I recognize my good fortune as Martin joined Jin as my co-supervisor. With his eye for detail, insistence on perfection, and measured caution, his inputs have greatly improved the quality of my work. I recall that in our first meeting with Martin as my supervisor, he insisted I call him by first name, saying that we were now colleagues. Both Jin and Martin have always made me feel that my thoughts are valuable, and that my words have worth. This perspective not only gave me confidence throughout my PhD but also made me realize that they are the kind of supportive mentor I would like to be in the future. I truly won the lottery with having both Jin and Martin as my supervisors, and all I can hope is that I have done their investment in me justice, and will continue to do so.

The Software Technology Lab has been the best environment that I could ask for. I am grateful to Mathieu, my soundboard to bounce off ideas, go-to for questions, listener when I needed to vent, and of course, my ice-cream buddy. I already miss the quick 2-minute breaks amidst serious work to laugh at a PhD meme together. Thank you to our "lab-regulars": Jazlyn, with whom I shared the nervous excitement of deadlines, making it easier to get past them; Avinash, for the passionate discussions on documentation design and for all his help both within and outside the lab environment; Divya and Bhagya, who became my unofficial guardians, always looking out for me on this PhD roller-coaster ride; Linh, who reminded me that life is to be enjoyed and not taken too seriously; Lanese, who never let me waver in my belief of myself; Sara, for her ever-caring always-listening nature; and Veronica for all those lovely baked treats which acted as rewards for internal milestones. To Fuyuan and Breandan, thank you for the regular reflections on research methods and graduate student life. Thank you all for being the motivation to take that 15-minute walk to get to lab, even on the days with the worst heat, heaviest rain, deepest snow, and unforgiving ice.

My thanks to the lab members, both present and past, whom I did not get to meet on a regular basis, but had the chance to interact with during lab events and meetings. Thank you to the research assistants with whom I worked: Kristen who supported the initial brainstorming phase of designing versatile documentation; and to Vivian and Shushi who contributed to the building of multimodal documentation. I also had immense support from friends and colleagues outside the lab. Thank you to Max, who has been a constant source of encouragement: the regular check-ins and uplifting words of “you can do this!”, made some difficult days much easier. Thank you to Arthur with whom I enjoyed discussions of research and life after PhD, through email correspondence and video calls. Thank you to Jessie, who not only provided valuable insights on Formal Concept Analysis, but also some lovely company over coffee. Thank you to Marco - it was amazing to meet someone outside our lab equally passionate about the state of learning resources for software.

Thank you to Andrew, Ron, and Corey who patiently solved any desktop, server, and resource related issues. I am deeply grateful to Ann, Kam, Diti, and Sheryl who were promptly responsive as I navigated the administrative processes of student life. Thank you to all the attendees at Bellairs 2020 and 2023, who made me feel welcome and comfortable in the world of research, and reminded me that what I had to say mattered. A special thank you to Gail, who took the time to discuss my PhD proposal and mentor me on how to reach for excellence. I am grateful to my PhD committee, Thesis Examiners, and Defense Committee who provided me with valuable feedback and guidance. Thank you to NSERC, who gave me the much-needed funding to conduct rigorous research.

I am supremely grateful to have met friends who became my extended family. To Arushi, Kshitij, and Anand, with whom I built my best memories, laughed the most, and had the craziest experiences - spending time with them seemed to make the stress of doing a PhD disappear. A huge thank you to Alex, with whom I could wind-down weekly over a good book and conversation at our weekly Buddy Read. Lots of love to Akshuz, who brought calm to any of my stormy feelings. Thank you to my dance fraternity; they helped me pursue my passion for the performing arts alongside my research.

And of course, thank you to my family. Thank you to my in-laws, Appa, and Athe who understood my drive for research and always encouraged me to prioritize my work. Thank you to Aishwarya who never hesitated to pamper me with gifts and cheer me up. Lots of love to my grandparents, Dada, Nanaji, and Nani, whose remarks were always along the lines of “work well, eat well, and stay healthy and happy”, and Dadi who I am sure is looking down at me with pride. I seek and cherish their blessings every day. My parents, Mumma and Papa - it is impossible to articulate what they have done for me. Everything I am and can do is because of them, and I will leave it at that. My unbounded love to Maitri, who, as always, has been a strong pillar for me to lean on. Experiencing adult student life, away from home, together, has brought us that much more closer, if it was even possible. I had heard that doing a PhD is difficult, but supporting a partner doing their PhD is also challenging. Vignesh has been the perfect partner, he celebrated my every high, no matter how small, and encouraged me through every low, no matter how shallow or deep. I can not imagine getting through life, let alone my PhD without him. And to my little Raghav, who showed me power in myself that I never knew existed, my endless, endless, endless love.

List of Figures

1.1	The top sources and online resources from which developers learn to code, based on the annual Stack Overflow Developer Surveys for 2022 to 2024 [183].	1
1.2	Examples of how software documentation resources can vary, even for the same technology.	2
1.3	An overview of documentation creation and information seeking for software-related information.	3
1.4	Overview of the four phases of our research, the corresponding study design, and the major contribution.	5
3.1	Diary entry template for each search session.	23
3.2	Process followed to obtain the data set for this study. The tables obtained in the last steps (represented by the last row in the figure), together form our data set.	25
3.3	The online software technology resource-seeking model of users learning a new technology. The components shaded in grey, i.e. QUESTIONS and RESOURCES, are posited components, while the rest emerged from our analysis. The numbers and asterisk annotated on the arrows indicate the cardinality of the relation. For example, multiple CUES can <i>be used to select</i> one RESOURCE, and multiple RESOURCES can <i>be accessed for</i> a single QUESTION.	28
3.4	Frequencies of the components and relations of the resource seeking model, in our data set.	38
3.5	Contingency table of <i>expands</i> relation between PREFERENCES and QUESTIONS. .	39
3.6	Adjusted Standardized Residuals and Contingency table for the <i>is accessed for</i> relation between RESOURCES and QUESTIONS. The values in each cell represent the frequency of connections between the pair of categories. Non-colored cells indicate that they were not included in the statistical analysis.	40
3.7	Adjusted Standardized Residuals and Contingency table for <i>is used to select</i> relation between RESOURCES and CUES. The values in each cell represent the frequency of connections between the pair of categories. Non-colored cells indicate that they were not included in the statistical analysis.	41
3.8	Contingency table of <i>is evaluated through</i> relation between RESOURCES and IMPRESSION FACTORS.	43

4.1	Example of identified blocks for the BeginnersBook resource “Constructors in Java”. We manually identified that the <code>article</code> tag inside the <code>main</code> tag with <code>class</code> “content”, contains the main content of the page. We treated each of these elements, such as the <code><p></code> and <code><pre></code> elements boxed in red, as an individual block.	52
4.2	Variation of resource properties by programming language. The red line indicates the median of the distribution.	56
4.3	Correlation between properties for significant relations in each programming language. Only the significant results ($\alpha = 9.5 \times 10^{-5}$; $p < \alpha$) are shown. The colors correspond to Pearson’s correlation coefficient values.	57
4.4	Distribution of the proportion of code blocks (from Table 4.3), against the average number of minutes per visit for the corresponding website.	58
4.5	Variations of extracted properties in Java Oracle, Java not-Oracle, and TypeScript TutorialKart resources. The red line indicates the median of the distribution.	65
5.1	Framework of documentors’ mindsets and their associated considerations across the three dimensions of the documentation contribution process, i.e. <i>motivations</i> , <i>topic selection techniques</i> , and <i>styling objectives</i> , based on interviews with 26 documentors.	72
5.2	Agreement responses of the 17 respondents to the validation questionnaire.	86
6.1	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.	93
6.2	The three code example modalities to demonstrate how to implement the character classes in regular expressions for Java.	94
6.3	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 6.1.2).	96
6.4	Optional open-ended questions in the survey.	97
6.5	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 6.1.2).	100
6.6	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 6.1.2).	101

List of Tables

3.1	Participant demographics for the diary study.	24
3.2	QUESTION categories.	29
3.3	PREFERENCE categories.	31
3.4	RESOURCE categories.	33
3.5	CUE categories.	35
3.6	IMPRESSION FACTOR categories.	37
3.7	Bonferroni-adjusted p-values calculated by Fisher’s Exact test using 200000 Monte Carlo simulations of connections between each pair of model components.	39
4.1	Details about the programming language and host website of the resources studied.	50
4.2	Properties extracted at the block level for each resource.	51
4.3	Computation of resource-level properties.	53
4.4	Mapping of <i>Less</i> or <i>More</i> of a property (from Table 4.3) to an attribute of a resource.	60
4.5	Distinguishing attributes (d.a.) in the <i>prominent styles</i> ($n=3$) for all resources.	61
4.6	Recurring Resource Styles in our data set for Java and Python resources.	64
5.1	Details of the informants of the interview study, all of whom are documentors.	71
5.2	Documentors’ considerations along the dimension <i>motivation</i>	74
5.3	Documentors’ considerations along the dimension <i>topic selection technique</i>	77
5.4	Documentors’ considerations along the dimension <i>styling objective</i>	79
5.5	Documentors’ mindsets and the corresponding considerations across the three dimensions of the documentation contribution process, for each informant.	81
6.1	Examples of the three programming task types in our survey.	95
6.2	Demographics of survey respondents.	97
6.3	Description of the 16 Fisher’s exact tests we performed. We conducted the tests between Dimension A and Dimension B, for each Filter.	98
A.1	Contents of the replication package [17].	150
B.1	Contents of the replication package (executable scripts) [18].	152
B.2	Contents of the replication package (dataset and results) [18].	153

C.1	Contents of the replication package [20].	156
D.1	Contents of Appendix D.	161

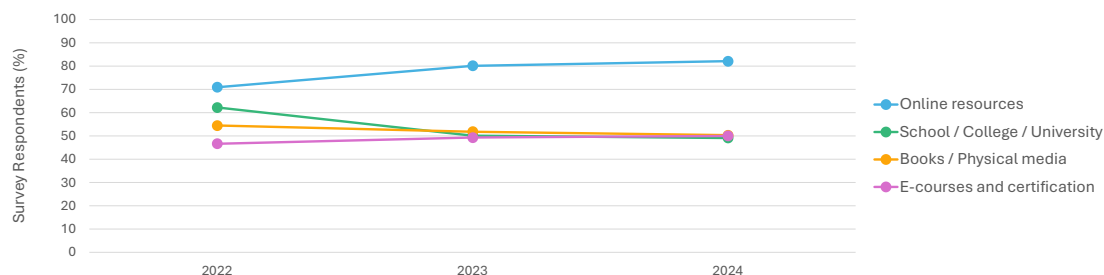
List of Acronyms

- **SE:** Software **E**ngineering
- **API:** Application **P**rogramming **I**nterface
- **HTML:** Hyper**T**ext **M**arkup **L**anguage
- **CSS:** Cascading **S**tyle **S**heets
- **FCA:** Formal **C**oncept **A**nalysis
- **URL:** Uniform **R**esource **L**ocator

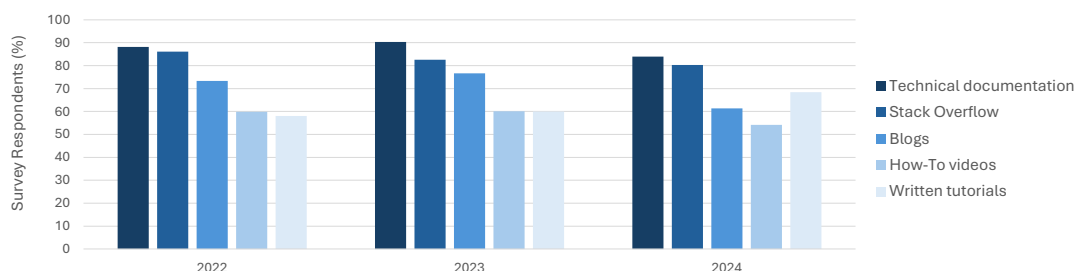
Chapter 1

Introduction

About 90% of programmers are partially self-taught in learning to code [182]. Thus, it is unsurprising that in just three years, there has been an increase from 70% to 82% in the percentage of programmers who use online resources to learn to code, as shown in Figure 1.1a [183]. Amongst the available online learning resources, Stack Overflow has reported that the most common resource that respondents of their annual survey used to learn to code was “technical documentation” (see Figure 1.1b) [183]. This statistic raises the question: what *is* technical software documentation?



(a) Top learning sources



(b) Top *online* learning sources

Figure 1.1: The top sources and online resources from which developers learn to code, based on the annual Stack Overflow Developer Surveys for 2022 to 2024 [183].

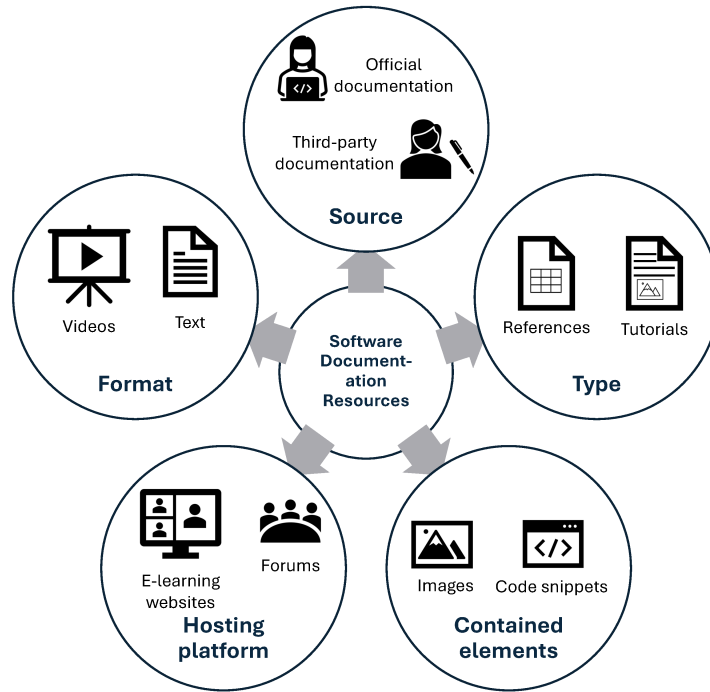


Figure 1.2: Examples of how software documentation resources can vary, even for the same technology.

Software documentation provides a unique opportunity to explain relevant technical information about a software technology. Such information can include how the technology works, how to use it, and even how to contribute to its development, maintenance, and improvement. Thus, for programmers, i.e. people who write code in any capacity, software documentation becomes an essential resource when using the technology, and has even been reported to boost developer productivity [179]. Consequently, the effective use of a software technology depends on the quality of its documentation. Usually, creators of a software technology release accompanying documentation, in which case it is referred to as *official* documentation. There also exist many websites dedicated to providing technology-related information. Enthusiastic technology users may also contribute documentation about a technology that they are not necessarily associated with, such as through technical blogs [187, 246] or video tutorials [96].

However, documentation creation is time-consuming [6] and tedious [228], because it is the process of collecting knowledge and ideas [35, 150] by converting tacit knowledge to connected, explicit pieces of information [208, 221]. Documentation creators are faced with a number of design decisions at different points during the software development process, including the format by which to present relevant information [54]. Additionally, they must consider the consequences of these design choices [24], and ensure that the information contained in the documentation is accurate and up-to-date [7].

As a result of these many design decisions, available software documentation varies in its structure, organization, and content [12]. Figure 1.2 shows examples of the many ways in

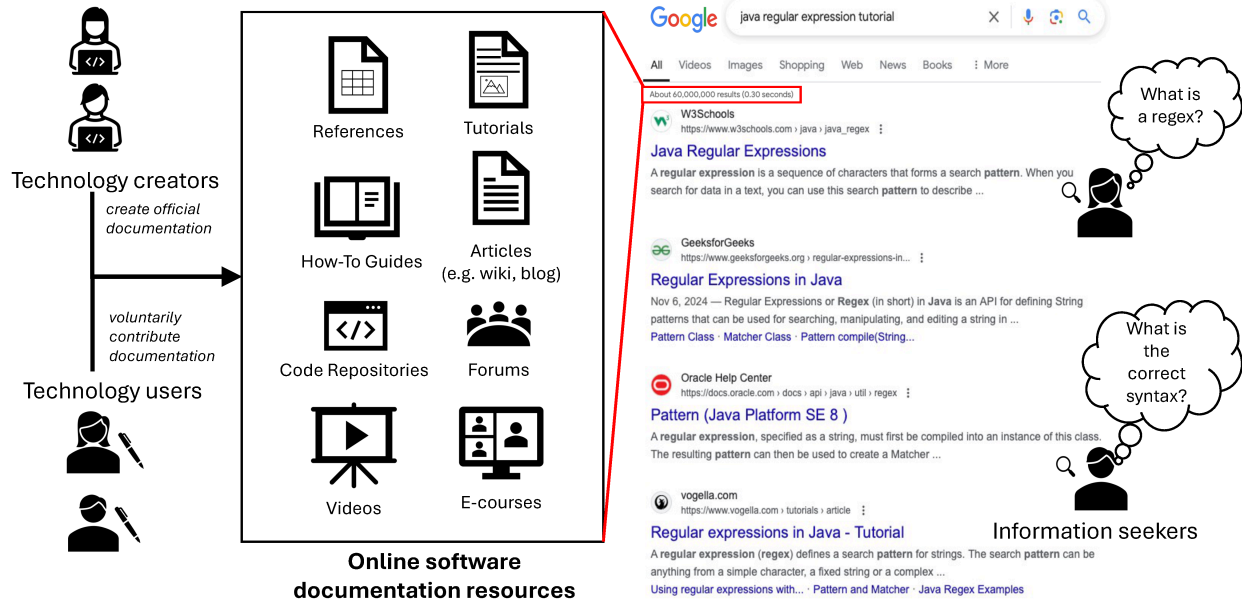


Figure 1.3: An overview of documentation creation and information seeking for software-related information.

which current software documentation varies. There are multiple *types* of documentation, including tutorials, user guides, and reference documentation. Each of the various types of documentation have been developed to cater to a different purpose, such that programmers can refer to the type that best suits their need. However, these purposes may not be easily distinguishable. For example, according to the Diátaxis documentation framework, both tutorials and how-to guides provide technology usage information, with the former focused on learning and the latter focused on completing tasks [198]. Additionally, a resource can have multiple elements such as overview information, code snippets, and advanced pages [260] as well as sections, links, and images [244]. Documentation may also be hosted on different commercial platforms (e.g. on Medium or Youtube). Recently, a number of portals for discussion and informal sharing of knowledge, including Q&A forums such as Stack Overflow, community repositories such as cplusplus.com and social platforms such as Discord, have also become sources of software technology information [201]. Although there are many frameworks (e.g. [41, 250]) and standards (e.g. [1, 192, 206]) that provide guidelines on how software documentation can and should be designed, they are not enforced and seldom used in practice. As a result, there is no consensus on what constitutes software documentation.

Programmers, then, must invest time and effort to find documentation that contains information pertinent to their needs, amidst the variety of documentation available [69, 166]. Consequently, programmers rely on search engines to navigate all available documentation [141]. Figure 1.3 demonstrates how search engines allow users to focus their search amongst the numerous available documentation resources contributed online. Still, a simple query of “java regular expression tutorial” on Google returns over *sixty million* web pages. As

a result, programmers must still *manually* browse through many documentation resources to find the information they need. Thus, the time taken to find relevant information can range from a few minutes to many hours over multiple search sessions. Often, programmers may even have to abandon their searches [124], despite having spent a significant amount of their time on the search, because of the common challenge of poor documentation [181]. Instead, programmers may look for alternative sources of information, such as reaching out to experts for help [185], or referring to source code directly to understand the underlying working of a software technology.

To support programmers in finding relevant information, prior work has focused on automatically recommending information from documents based on user queries [111, 191]. In the latest advancements, the combination of machine learning models, generative artificial intelligence, and chatbots have produced tools such as ChatGPT that allow users to retrieve customized information. However, such technologies may be sources of inaccurate information, prompting many users to be hesitant of trusting information gathered and presented through artificial intelligence (AI) [115]. Furthermore, these techniques have focused on *retrieving* information, and still rely on the quality of programmers' search queries. However, the information seeking process relies on the information *needs* of a programmer, which can impact other aspects of documentation search, such as the source that they prefer to gather information from [114]. Thus, it is important for a user to remain an integral part of the search process, granting them control of how to navigate through results, as they search for the information based on their context [28].

Effective software documentation is thus documentation that caters to varied needs, while allowing programmers to control their browsing behaviour. Designing such versatile documentation is nontrivial, as it involves many considerations from multiple perspectives. Whereas programmers are the target audience for such documentation, it is also important to consider the effort and thought processes that go into *creating* documentation. Despite the synergy between documentation creation and information seeking, there is ambiguity about how the creation and information seeking processes interact through the medium of software documentation. For example, it is unclear how documentation creators consider the various needs of potential audiences. What are the resulting variations visible in documentation, and do they sufficiently address programmers' information needs about a technology?

Although prior research has studied the design of software documentation [56, 106, 239, 244], there is a disconnect between what is learned about the information seeking behaviour, and work on documentation generation. Whereas, research on the former has focused on human behaviour, prior work on the latter has focused primarily on automated methods to gather and present information. Consequently, the critical human aspect of creative input during documentation creation and context-specific needs during information seeking is discounted during documentation design. For this thesis, we investigated the complex interactions of documentation creation and information seeking through documentation design. We focused on the two human perspectives of the design of software documentation: the point of view of the *documentation creator*, and the point of view of the programmer who is the *information seeker*. We also investigated the variations in design of available documentation.

	Study Design	Primary contribution
Phase 1: How Programmers Find Software Documentation	Diary and interview study with ten programmers learning a new technology	Theoretical model of programmers' resource seeking process
Phase 2: Documentation Properties and Styles	Data mining of 2551 software tutorials across 22 websites for five programming languages	Framework to characterize tutorials based on their <i>style</i>
Phase 3: Considerations of Documentation Creators	Interview study with 26 documentation contributors	Framework of mindsets and considerations of contributors
Phase 4: Interactions with Multimodal Documentation	Survey to complete tasks using multimodal documentation with 55 programmers	Development and evaluation of a multimodal documentation prototype

Figure 1.4: Overview of the four phases of our research, the corresponding study design, and the major contribution.

Overall, we investigated **how documentation can be designed to cater to the various needs of information seekers, while considering the efforts of documentation creators**. We employed empirical research methods to observe documentation creation and information seeking practices. The empirical techniques involve the collection of evidence, for example through user studies with human participants or mining of online data, that can support the formation of conclusions about human and non-human aspects of software documentation. Based on our data-driven approaches, we developed frameworks to capture how people interact with software documentation. Additionally, we presented an overview of the design of current documentation resources. Our research proceeded in four phases, with each phase focusing on one perspective of the creation, design, and use of software documentation, culminating in the creation of a documentation prototype, as shown in Figure 1.4.

Phase One: How Programmers Find Software Documentation

First, we studied how people *find* relevant software documentation. Prior work has reported that people primarily use an *orienteering* information seeking strategy [241]. Orienteering refers to using small steps to move towards a perceived destination that contains the information needed. This strategy may be preferred because of its cognitive ease, allowing people to be in control of their location and of context during their search. However, it is unclear how programmers decide their next step and which resources to access, in the orienteering strategy. Thus, we wanted to understand the *thought process* behind the decision-making process of navigating between different documentation resources about the same topic.

We investigated the question of **how to represent the behaviour and rationale of programmers as they search for online resources when learning a new technology**.

We conducted a diary study with ten programmers to closely examine the kinds of resources they referred to for different queries and how they navigated to those resources. From our analysis of the diary entries, we proposed a *resource-seeking model* to characterize the process of finding online software documentation learning resources.

Our model is comprised of two groups of components: *Need-oriented* and *Resource-oriented*. Need-oriented components are QUESTIONS, PREFERENCES, and BELIEFS. These components of the model capture information or context about a programmer’s information need. Resource-oriented components are related to the resources that programmers access and include RESOURCES, CUES, and IMPRESSION FACTORS. Our model describes nine relations between the six individual components, for example, that a CUE *is used to select* a RESOURCE.

The resource-seeking model surfaces the formerly implicit components that guide the search for technical information. Awareness and understanding of the interplay between these components is helpful for information seekers to effectively form search queries, and to documentation designers to facilitate the seeking process by presenting information purposefully.

Phase Two: Documentation Properties and Styles

Whereas the first phase provided insight about what programmers want, need, and prefer of software documentation resources, we also wanted to determine to what extent existing online documentation caters to these diverse needs and preferences. In the second phase, we studied the design of programming tutorials in the current documentation landscape. We investigated the research question **to what extent do software technology tutorials vary in their structure and content-related properties?** Based on observations of our investigation into these properties, we explored the question **how can we systematically reason about the design of software technology tutorials?**

The study followed a data mining research method to extract and analyse design properties of programming tutorials. We focused on tutorials for Java, C#, Python, Javascript, and Typescript, and refer to each tutorial page as a separate information *resource*. We extracted properties for 2551 popular resources, such as the number of code fragments present and depth of sectioning of the content. We contribute a detailed analysis of the properties of software documentation resources organized by programming language. Based on our observations, we proposed a framework to characterize a resource’s *resource style* as its combination of *distinguishing attributes*, i.e. properties that vary from the norm. Our conceptual framework supports three techniques for identifying resource styles, namely *prominent styles*, *recurring styles*, and *user-defined styles*.

Our framework for characterizing resources provides insight to documentation creators about the design of resources for software development technologies to make future design decisions in a systematic manner. Our observations also provide a means to help information seekers systematically identify pertinent properties when comparing documentation resources for the same technology.

Phase Three: Considerations of Documentation Creators

To understand why there is so much variation in the design of documentation resources, it is necessary to gain insight on how documentation is created, and with what purpose. In the third phase, we investigated the research question: **why and how do people voluntarily contribute software documentation online?** We interviewed 26 documentation contributors to understand why they volunteered their time and effort to create software documentation when there may already exist other online sources of information on the same topic. We asked informants about why they began contributing documentation, as well as how they went about the creation process. This included, for example, how they determined what technologies and topics to cover and how they made decisions about the style of the documentation. We performed qualitative analysis of the interviews, from which we elicited sixteen *considerations* documentation contributors have, across three major *dimensions* of the documentation contribution process, i.e. *motivations*, *topic selection techniques*, and *styling objectives*.

We noted that considerations across different dimensions are thematically related. We grouped related considerations and refer to the groups as *mindsets*. A mindset describes a particular combination of motivations, topic selection techniques, and styling objectives that captures the thought process of the documentation contributor. For example, some informants were motivated to create documentation because of *inadequate documentation*. To select topics, some informants chose ones for which documentation did not exist before, *to fill the documentation gap*. Some informants thought about how *to differ from existing documentation*, when styling their content. From these three considerations, we elicited the mindset: *novelty and value addition*.

The mindsets and their associated considerations provide a framework for characterizing the documentation contribution process. This framework provides the documentation creation context, which can surface cues for information seekers to use as they search for pertinent resources. For example, knowing that a documentation contributor has the *novelty and value addition* mindset, indicates that their documentation covers information about the technology in an alternate, novel manner. An information seeker struggling to learn a technology can identify that this contributor's documentation will be relevant to them, rather than documentation of another contributor who predominantly has the *growth and visibility* mindset. Similarly, our insights can inform the design of documentation tools to support documentation contribution.

Phase Four: Interactions with Multimodal Documentation

From our investigation on how people find learning resources online, we noted that programmers have different pre-existing preferences about the content or style of resources [16]. For example, some developers refer to code in order to duplicate it [124], and thus may prefer *complete* code examples that can be executed [24]. Other developers find small code examples focused on patterns of usage useful [210]. 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 [7]. Thus, creating docu-

mentation must account for the audience and their varied preferences, a consideration that we noted documentation creators have, from the previous study phase.

In the fourth phase, we proposed the idea of *multimodal documentation*. A multimodal resource contains information through multiple *modalities* that are inspired by diverse presentation formats including variations in information conciseness. Thus, multimodal documentation allows users to select information according to their presentation preferences. We designed a prototype tutorial containing five modalities, i.e. text, tables, and code examples in three different modalities (regular, summarized, and annotated). We investigated the research question: **how do programmers make decisions about their presentation needs and preferences in a programming tutorial?**

To answer our research question, we created three multimodal tutorials about three basic programming concepts in the Java language, namely 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 use one of the multimodal tutorials to complete three different programming tasks related to that topic. Each of the three tasks were of a different *task type*, 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 observed how respondents used the different modalities for the different task types. We support our findings with statistical analysis of the responses and insights from the open text responses in the survey. We found that, irrespective of the topic, for conceptual 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 indicate 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.

Contributions from the Four Phases

Our research across the four phases provides insight on software documentation design possibilities that can cater to programmers’ needs while maintaining creators’ considerations. The contributions of this thesis include a model for how programmers search for documentation online, a framework for characterizing documentation based on their varying properties, and a framework of contributors’ mindsets and their associated considerations when creating and contributing documentation (see Figure 1.4). In addition to the theoretical models arising from this work, we contribute an overview of the design variations of software tutorials. We also contribute a prototype resource for presenting information to cater to differing user needs, as well as an empirically-backed description of the usefulness of its different presentation formats for different contexts. Our work can be leveraged to assist programmers in

seeking pertinent information about a technology in an efficient manner. This work also generates insights for documentation creators to understand their target population's search behaviour and how the documentation they create can satisfy programmers' preferences and needs. Our findings take a step towards designing personalized documentation [211], while retaining users in control of their information seeking process, and minimizing additional effort for documentation creators.

1.1 Thesis Organization

The remainder of this thesis is organized as follows. Chapter 2 develops the background and establishes the related work for the four phases of the research. Chapters 3 to 6 describe each of the four phases of our research. In each phase-related chapter, we provide a brief introduction, followed by an overview of the *goal*, the *research question*, as well as the associated *publication* and *replication package* for the study.

Chapter 3 describes the first phase of our research, regarding how programmers search for learning resources online. We provide an overview of the resource seeking model we developed from our study (Section 3.2), so as to provide context when describing the research design and goals. We then describe our study design, including how we conducted the diary study, and analysed the diary entries (Section 3.1). We present the different components of the resource seeking model (Sections 3.3 and 3.4), and the relations between them (Section 3.5). Finally, we discuss the implications of our findings (Section 3.6).

Chapter 4 describes the second phase of our research, related to the properties and styles of available software tutorials. We describe how we collected the tutorials, and extracted the properties for analysis (Section 4.1), and present the variations in tutorial properties (Section 4.2). We then describe how the properties can be used to characterize resources as being of a particular *style* (Section 4.3).

Chapter 5 describes the third phase of our research, about why and how programmers contribute software documentation. We begin by introducing our study design including how we recruited informants and conducted the interviews (Section 5.1). We describe the considerations informants had across the dimensions of the contribution process (Section 5.2) and the mindsets that correspond to combinations of the considerations (Section 5.3). We end the chapter with a discussion about the validation of our results and the implications of our findings (Section 5.4).

Chapter 6 describes the fourth phase of our research, introducing the design of a multi-modal documentation, and our evaluation of how programmers interact with it. We explain how we designed the study (Section 6.1) and the results of our user survey regarding the multimodal documentation prototype (Section 6.2).

Chapter 7 presents a discussion based on our results from each of the four phases. We describe insights on software documentation design (Section 7.1), anticipated challenges for designing customizable documentation (Section 7.2), and future work in this regard (Section 7.3). Finally, we conclude this thesis with an overview of the research, its contributions, and the most important takeaways in Section 8.

Chapter 2

Background and Related Work

A software technology can only be used effectively if programmers are able to understand how to leverage its features appropriately. Thus, software technology documentation plays an important role in connecting software creators to technology users, such as programmers. Despite this critical need for software documentation, it is often neglected, because creating documentation is a time and effort intensive process [54]. As a result, current software documentation has multiple issues, including poor readability or incompleteness [7]. Prior work has investigated the ability to automatically generate documentation, for example from source code, to alleviate the burden on technology creators [91, 186, 235]. Furthermore, research has investigated the information that programmers *need* when performing software engineering activities [114, 123, 165, 229], and provided guidelines on what documentation should contain [56]. Still, documentation can vary in its content [15] and presentation [24], resulting in numerous documentation resources available online. Consequently, programmers can be overwhelmed by the amount of online content, which makes finding pertinent information a time-consuming task.

Priestley suggested the idea of a *dynamically assembled document* that would allow users to restructure the information present in the document into multiple views based on their needs and preferences [196]. Robillard et al. proposed *on-demand developer documentation*, an automatically generated document built based on knowledge of task context and users' needs [211]. In both visions of documentation, a critical aspect is that the presentation and organization of the documentation must cater to users' preferences. However, designing such a document that can cater to a variety of needs and preferences is non-trivial. It is first integral to understand how these needs and preferences manifest, and how they are currently catered to.

We contribute to existing research on the design of software documentation. We study documentation from the perspective of documentation creators and information seekers, as well as investigate how documentation can be designed based on these perspectives. Thus, we review prior literature on the *documentation seeking* process, the *documentation contribution* process, and the *design of documentation*, and relate them to the different phases of our research.

2.1 Documentation Seeking

Understanding how technology users search for information that is relevant to their context can provide insight into what they may look for in documentation to identify it as “pertinent”. Literature on information seeking finds its roots in food-foraging behaviour of people and animals [126, 263]. The earliest *information* foraging model, suggested by Pirolli and Card, is decomposed into three sequential theoretical models, i.e. the patch model, scent model, and diet model [194], inspired by food-foraging theories. Pirolli and Card described that the information that can solve users’ needs may be contained in multiple information “patches”. To find the information they need, users perform “scent-following” wherein they use cues or hints to decide how much time to allocate to moving between or within patches. The diet model describes how users select, make sense of, and consume information related to their needs.

The first phase of our research focuses on investigating the “scents” that programmers follow to make decisions about which “patch” of information, i.e. learning resource, could be pertinent to their information needs. Our work builds on two posited components of information seeking: the information need, and the documentation resource. We studied the behaviour of programmers in decision making from posing a QUESTION to accessing a pertinent RESOURCE.^(a) In the fourth phase of our research, we gained additional insight into the preferences for information presentation that programmers had for different programming contexts. We discuss related literature on understanding the *information needs* that programmers have, their *resource design preferences*, their *online resource seeking* behaviour, and how they *identify pertinent information* to resolve their needs.

2.1.1 Information Needs

In the context of software engineering, prior work has mainly focused on the needs of developers in every-day development and maintenance activities. Based on their experience, Erdos and Sneed suggested that there are seven questions that a programmer must have the answer to, to be able to maintain a software program [67]. Sillito et al. found that when changing software, programmers ask questions that can be grouped into four categories related to knowledge about the entity graph of code components [229]. These works primarily focus on code behaviour (e.g. *Where is a particular variable declared?* [67]), whereas the QUESTIONS we observed in our study cover a broader scope, including any aspect that a programmer may want to know about when *learning* the technology, such as underlying concepts or installation-related questions (see Section 3.3.1).

Ko et al. performed an observational study to determine what kind of information developers generally look for, and categorized search instances into 21 information needs [123]. Gallardo-Valencia and Sim asked 25 developers from a company to self-report over a period of 15 days their web searches [77]. They found five main types of problems that induce searching for information online. Duala-Ekoko and Robillard performed a study in which

^(a)We use the term *resource*, instead of *documentation* for this phase as programmers accessed some web pages that can not be clearly classified as being a traditional form of documentation.

twenty participants were asked to think aloud as they worked on two programming tasks in a familiar programming language but using unfamiliar Application Programming Interfaces (API)s [60]. The authors analysed the screen recordings with the verbalized thought process and observed that the participants had twenty types of questions. Rao et al. studied users' web search behavior for software engineering tasks by analyzing the logs of millions of search queries to the search engine Bing, and found six categories of intent for searches [204], including searching for conceptual information. We revisit these intents to evaluate the multimodal documentation in the fourth phase of our research (see Section 6.1).

Some overlap exists between the work on information needs and categories of the QUESTION component in our study on how programmers find online resources. For example, *In what situations does this failure occur?* [123], *the existence of errors in software* [77] and *Debug* [204] correspond to the *Debug* category in our work (see Section 3.3.1). Despite this strong correspondence, we chose not to reuse previous categorization. We focus on the questions of *programmers* who are *learning* a new technology. Although the categories may be similar, the context while searching for this information differs from everyday information look-up. Hence, we did not want to assume that prior taxonomies would completely encompass questions by programmers who are learning a technology, or alternatively that the programmers would have questions in all existing categories.

Erdem et al. recognized that questions are composed of multiple factors, and proposed a model to represent the questions that programmers have while trying to understand software [66]. The model identifies a question by three components - its topic (the subject of the question), its question type (e.g. *who*, *what*, *where*), and its relation type (i.e. the kind of information that is requested). In addition to the questions programmers learning a new technology had, we studied the requirements the programmer had about the information they were seeking (PREFERENCES), and why these requirements existed (BELIEFS) (see Section 3.3).

2.1.2 Resource Design Preferences

In addition to having 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 [69]. 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 in the final phase of the research, 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 (see Section 6.2). 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. re-

ported 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 [62]. 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 [143] that the role of “software engineer” involves [161], and the variations of modality use based on the programming task types (see Section 6.2), indicates the need to have documentation that can be organized in a flexible manner.

2.1.3 Online Resource Seeking

Teevan et al. identified that participants in their user study followed an *orienteering* strategy, i.e. using small steps to navigate as they attempt to find the information they need [241]. The researchers proposed that search engines should support the orienteering strategy, for example by providing meta-information, cues, and context of search results to inform users. To gain insight into scent-following behaviour, Pirolli and Fu measured information scent of a web page as the mutual relevance of its contents [195]. They found that it was useful in foretelling user actions such as from which page a user will leave a website.

Brandt et al. studied *why* programmers search for information by performing an in-lab study with 20 programmers [34]. They observed that programmers searched online to clarify existing knowledge, remind themselves of details, or to learn by trying code snippets. In the latter case, participants used primarily aesthetic aspects, such as the existence of advertisements on the web page, to quickly judge whether to read through the page. The authors also analysed a web query data sample containing 101,289 queries from 24,293 programmers, to gain a deeper insight into the search process. They found an association between the types of pages visited and the type of queries performed. For example, they found that code-only queries resulted in more API documentation accesses, and natural language queries to more tutorial accesses. We also performed statistical tests to determine the association between types of questions and resources accessed (see Section 3.5). However, our categories of QUESTIONS revolve around the content of the question, as opposed to Brandt et al.’s study which focuses on the format of the search query.

2.1.4 Identifying Pertinent Information

Prior work has studied the seeking behavior for code within a target resource. Lawrance et al. proposed the Programmer Flow by Information Scent (PFIS), an algorithm to describe how programmers navigate through source code during debugging based on a bug report [134]. This algorithm involves measuring the “proximity” of each possible area of source code the programmer could go to (e.g. package, class, method, or variable) with the bug report content, and calculating the probability of a visit based on multiple simulations

of traversal. Lawrence et al. compared the algorithm’s prediction of which piece of code will be visited to observed human behaviour and found that PFIS predicted human navigation close to aggregated human decisions. Ragavan et al. also focused on navigation between code artifacts by studying how programmers compare similar pieces of code to determine which one is applicable to a particular task [234].

Piorkowski et al. studied how the *intent* of a developer, i.e. to fix a bug or to learn to help someone else fix a bug, affects the type of information sought. They performed a user study with eleven participants, split into two equal sized groups with tasks of the two different intents [193]. The authors observed that there was a large overlap between *cues* that participants with both intentions used to navigate the package explorer, editor, stack trace, and search results during debugging. They observed that a majority of cues used were code output or domain related. Liu et al. performed a formative study with fifteen programmers to understand how they reuse programming decisions regarding technologies used in particular scenarios, made by other programmers [146]. Liu et al. elicited three major *facets* that help assess whether reuse is appropriate in programming: *context of the prior decision*, *trustworthiness of the web source and the author*, and the *thoroughness of the knowledge for reuse*. In contrast, our work in the first phase of research focuses on programmers’ rationale when navigating available online resources for information about a technology, irrespective of the particular format.

Nadi and Treude studied the navigational cues in finding relevant answers in a Stack Overflow post. They reported that *essential sentences*, i.e. ones which users can use to determine whether an answer is worth reading or not, highlighted by most participants mainly contained explanations, or specified a library or a code component [172]. Marques et al. determined that sentences within an artifact, perceived by their participants as relevant to a task, contained common semantic meanings that could help determine what information within the artifact is relevant [158]. They also observed that while participants used different search strategies, they used implicit clues to find the information they needed. For example, they would judge the value of text based on visual cues like whether it was in bold, or was concise.

Sadowski et al. performed a case study at Google, via a survey conducted every time a participant accessed their internal search website, to gain insight on why and how programmers at the company performed searches [217]. They augmented this study method with an analysis of log data to determine quantitative measures of search session, such as how many terms were in queries or the average number of clicks that lead to a successful search. They reported micropatterns of observed search sessions. For example, they found that programmers who are *very familiar* with code typically follow the micropattern: one or more searches followed by one or more clicks. Bai et al. performed a follow-up task-oriented lab study with graduate students [26] and compared their results with the observations of Sadowski et al. [217]. We complement prior literature on within-artifact information seeking literature. In the first phase of our research, we focused on the decisions that programmers make in choosing between different artifacts in the specific context of learning a new technology. In the final phase, we gained further insight on how programmers made decisions about accessing information presented through different modalities within a single resource.

2.2 Documentation Contribution

The context in which documentation was created can provide insight into why it is organized, structured, and presented a particular way. For example, peoples' motivations affect their creativity [72], knowledge sharing [103], and contribution to open source software [80]. These findings suggest that a documentors' thought process can have an impact on the documentation they create. Furthermore, understanding why and how a documentor makes many of the design decisions that arise during documentation creation [24] can provide insight on the purpose and intended use of the created documentation.

In the third phase of our research, we explored why and how people voluntarily create and contribute documentation. We discuss related prior research on *motivation to create documentation*, *documentation creation practices*, and *mindsets in software engineering*.

2.2.1 Documentation Motivations

Ryan and Deci proposed the self-determination theory (SDT), which introduced a taxonomy of motivation including amotivation, intrinsic, and extrinsic motivation [216]. Personal blogs have been found to be sources of therapeutic reflection and experience sharing [81, 271]. Li elicited seven reasons why adults blog, including for *self-expression* and for *socialization*, from a questionnaire filled by 288 bloggers [140]. In the context of software documentation, Shmerlin et al. conducted interviews with five software developers and a questionnaire with ten developers to understand the motivations of developers to document their code [228]. Their participants indicated increased code comprehensibility, structure, and quality as what they enjoyed most about documenting, while acknowledging that it is difficult and time-intensive. McArthur discussed four common prejudices against documentation, one of which is that programmers would rather program than write documentation [159].

From a survey with thirty bloggers, Parnin et al. reported four types of technical blogging motivations [189], including for personal branding and as a personal knowledge repository. MacLeod et al. studied screencast documentation wherein developers record their screen and explain how the corresponding technology works [155]. They analyzed 20 Youtube videos and interviewed ten screencast creators and reported five reasons *why* developers create the screencasts, which included to build an online identity, and to promote themselves. Whereas prior literature has focused on developers' motivations to create either text or video content, our informants include both text and video documentation creators who are not necessarily developers. Additionally, as part of the interview, we encouraged informants to elaborate in detail about how they *began* contributing documentation. We found that the motivations we elicited correspond to the aggregate of those reported by Parnin et al. and Macleod et al., and additionally introduce the novel motivation consideration *related pursuits* (see Section 5.2.1).

2.2.2 Documentation Creation Practices

Bottomley performed interviews with 24 documentation writers to understand their skills and practices [33]. They found that documentation creation requires skills such as recognizing unclear instructions, understanding high-level concepts, and predicting and responding to user needs. Schmidt proposed an analytical framework for technical blogging that has three structural dimensions: *rules* about the appropriateness and medium of content, *relations* such as hyperlinks to other resources, and the *code* in the blog [220].

Dagenais and Robillard interviewed 22 developers and technical writers who either wrote or read documentation to understand how documentation evolves [54]. Additionally, they manually inspected the evolution of 19 documents (over 1500 revisions) from 10 open source projects. They reported different decisions that documentation involves, such as determining what kind of documentation, i.e. *getting started* or reference documentation, should be written first. The authors also reported that one way to maintain documentation alongside the evolution of a project is to document the change immediately. Wang et al. reported how the nine identified categories of documentation in 80 popular computational notebooks mapped to stages in the data science life cycle [255]. Prior work has elicited how to write articles such that they motivate the reader to read instructions [118, 148]. For example, Goodwin provided suggestions for technical writers on how to style manuals such that it “emplots”, i.e. involves the reader in an action-oriented storyline [85]. Similarly, different information styles, such as *declarative* and *procedural* information, have varying effects on the instruction-following behaviour of readers [154, 248]. Guidelines exist for designing documentation [198] and technical writing [32], and techniques exist to support the automatic generation of documentation [91, 186, 254, 267]. Understanding the thought process of documentors can inform the design of such tools.

From a mixed-methods study involving surveys with over 150 developers, and eleven semi-structured interviews, LaToza et al. reported that knowledge about software projects, for example the architecture and design rationale, is not systematically documented, and instead, much of this information is only “in peoples’ heads” [133]. However, the majority of their participants agreed that understanding the rationale behind code, though important, was challenging, partially because this information was never documented. As a result, there is a need to better support the capture of mental models and design rationale, in order to support decision making [92] in programming tasks and future stakeholders’ understanding of a project.

2.2.3 Mindsets in Software Engineering

There exist theories in psychology and education research domains regarding *mindsets*, e.g. growth versus fixed mindsets [29] and mindsets related to phases of action [84]. The study of mindsets is very useful in providing appropriate recommendations to users, for example, to points of interest in a city [253]. In software engineering, the term mindset is associated with engineering practices or tasks. For example, the Agile mindset is a common terminology used to describe the principles required to practice agile software development, e.g. flexibly

adapting to change instead of rigorous prior planning [215]. To *define* what the Agile mindset involved, Mordi and Schoop analysed 23 papers published in research venues or privately by practitioners, and performed interviews with 17 industry practitioners [169]. From these sources, they derived 27 characteristics to describe the “Agile Mindset”. For example, they reported that all sources had evidence of *trust* and *responsibility and ownership* as characteristics of the mindset. Motogna et al. conducted a study with 47 student teams in a 14-week long course, in which students were encouraged to follow Agile practices [170]. The authors reported that developing appropriate soft skills associated with such characteristics is important to help adopt the mindset needed.

Software engineering research has also focused on the *privacy mindset*, which describes the thoughts of people as they navigate privacy features and invasions in online websites [89, 107]. Similarly, the exploration of the *security mindset* describes how people think about security in source code and software [237]. To understand security and privacy mindsets, Arizon-Peretz et al. conducted interviews with 27 practitioners and analyzed their responses, in the context of themes from organizational climate theory, i.e. factors of the working environment [13]. Despite extensive research on mindsets, there is no common definition for the term *mindset* [36], as it depends on the perspective and context of the related research. In phase three of our research, we identified documentor *mindsets*, which capture the implicit relations between considerations across the different dimensions of the documentation creation process, based on interviews with 26 documentation contributors (see Section 5.3).

2.3 Design of Documentation

Given that there are many different contexts for which software documentation may be accessed, as well as the variety of design choices to create documentation, there is a need for versatile, customizable documentation. Prior research has concentrated on the information content of software documents [45, 152] and the style of the information presented [119, 248]. In prior work, we reported that there are multiple decisions related to structure and content, that impact the design of a software tutorial [24]. Fourney and Terry emphasised that tutorial content must be formalized, because varying communication techniques pose a challenge to machine understanding and generation of tutorials [74]. Mehlenbacher elicited that documentation development involves establishing *design goals* to ensure usability of documentation [163]. Prior research has focused on adapting documents to suit information needs as described in Section 2.2.2, and to help users locate relevant information efficiently [8, 111, 149, 242]. To support resource creation and resource seeking, we investigated how to systematically reason about the design of software tutorial resources and proposed multimodal documentation to cater to varied resource preferences.

In the second phase of our research, we focused on software tutorials, a particular type of documentation [198]. We studied how software tutorials varied in their design properties, and how we could identify them as being of a particular style. In the fourth phase of our research, we explored the ability to present the same information through multiple *modalities*, or information presentation formats, within a *multimodal documentation* resource. Programmers

seeking information, thus, will have the opportunity to select amongst multiple modalities based on their preferences and the context of their search, thereby granting them control of their search process.

Our work is related to prior literature on the *characteristics of documentation*. We also discuss previous work that applies *Formal Concept Analysis*, the technique which we relied on to identify *recurring styles* in Section 4.3.2. Finally, we discuss prior work related to *user controls of navigating documentation* during information search.

2.3.1 Characteristics of Documentation

Angelini studied the API reference documentation of eight web applications and reported variations in the way information was presented, e.g. in some cases, a separate section was dedicated to the syntax or description of an API, and in others, the information was not clearly labeled [12]. Tiarks and Maalej performed an exploratory study of 1274 tutorials on Android, Apple iOS, and Windows Phone OS to understand the nuances in mobile app development tutorials [244]. In prior work, we studied the design of three Android tutorials from different sources [24]. In both studies, the researchers reported how tutorials varied in structure and content. In our study, we also provided a set of guidelines for thinking about the consequences of different design decisions [24]. Head et al. analysed the structure of code snippets in 200 online tutorials to inform the creation of Torii, a tool to generate tutorials from linked source code [91].

Tang and Nadi developed and evaluated a tool to summarize nine metrics related to the quality of documentation including readability and ease of use [239]. Despite methods to evaluate the quality of documentation [8], there is no consensus about how documentation should be designed. The Diátaxis documentation framework consists of four structural modes, *tutorials*, *how-to guides*, *reference*, and *explanation* and provides guidelines for the overall content and styling of each mode [198]. Other research focuses on specific low-level aspects such as manually identifying what kind of information about code snippets can be extracted from software documents [45]. Although prior work elicits design implications based on difficulties with learning software [210] and guidelines to design documentation [167], these studies focus on documentation for Application Programming Interfaces (APIs). In phase two of our research, we used a semi-automated, data-driven approach to characterize tutorials based on their design.

Dagenais and Robillard defined *documentation patterns* as *coherent sets of code elements that are documented together* [55]. They proposed AdDoc, an automated method to capture these patterns in the documentation of frameworks. In a previous study, we observed that between 11% and 56% of sentences that provided *API information* in a sample of tutorials for Java and Python could be replaced by their API reference documentation counterparts [15]. We proposed an *information reuse pattern* to support such systematic reuse of information between the two documentation types.

Researchers have proposed methods to document frameworks [41, 113]. Butler et al. proposed a *reuse case* which documents the reuse of a framework [40]. A reuse case categorizes the type of documentation that is used in a particular *category* of framework reuse, e.g.

while *selecting* a framework, and the *aspects* of reuse, e.g. the granularity in terms of class methods. In contrast, from the second phase of our research, we propose the characterization of resources based on their *styles* and support the grouping of resources based on recurring styles (see Section 4.3).

2.3.2 Formal Concept Analysis

Formal Concept Analysis (FCA) is a mathematical technique to identify *concepts* as a set of objects and their common attributes (see Section 4.3) [79]. In the context of software, FCA has been used to model common and uncommon features of different software product variants [9, 10, 75, 76]. Prior work has investigated the ability to characterize code by the relations between classes, to support code analysis [83] and to identify design patterns [37]. As a result of literature surveys conducted by Tilley et al. [245] and Ferré et al. [71], and an investigation into FCA for data mining by Valtchev et al. [249], the authors emphasise the potential of FCA for knowledge discovery in a variety of domains, including software engineering. We leveraged FCA to elicit commonly co-occurring combinations of resource properties as *recurring resource styles* (see Section 4.3.2).

2.3.3 User Controls for Navigating Documentation

To find relevant information, developers use different strategies, and leverage their knowledge about where to look for information [141]. Software developers may also use *cues* such as *creation time* and *update time*, when searching among multiple similar source code snippets [200], or judge the potential value of text based on its styling to find the information they need [158]. Prior work has studied how to support information seeking with explicit cues for users, such as an indicator of the time cost of reading a resource [112], providing an overview of all comments in a blog conversation [95] or of all pages in a document [88].

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 [38, 65, 128, 162, 273]. Käki and Aula [116], and Käki [117] 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. The researchers reported that the categories were especially useful when search engines were unable to retrieve relevant results due to general, vague search queries.

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 [147]. The authors conducted a survey with four tasks to determine developers' preferences for either view. They reported that for all four tasks, more than 50% of participants preferred inline views with an optional separate panel. 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 from the fourth phase of our research, we also report

contradictory modality preferences between respondents (see Section 6.2.6), 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 [5]. 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. These observations show promise for user-controlled customizable software documentation, allowing technology users to manipulate a resource to their needs and preferences.

To cater to varying needs, documentation creators are forced to manage multiple formats of software documentation [61] to avoid information inconsistency [7, 15]. 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 [190]. It is no surprise then, that software documentation creation can be a tiresome process [6]. Instead, our findings from the entirety of our research 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.

Chapter 3

How Programmers Find Software Documentation

The effort required in manually filtering resources during navigation may be attributed to the fact that there are many aspects that information seekers consider. Information foraging theory suggests that users typically follow *information scents* that help identify the information most suited to their needs [194]. A *scent* refers to the perception of the value of information given by cues, such as citations or ratings. Information scents have proved useful in foretelling user actions such as when a link to a web page is followed or from which page a user will leave a website [195]. To gain a deeper understanding of this behaviour for programmers seeking information online, it is necessary to observe their resource seeking behaviour, and identify the rationale behind their navigation decisions.

We conducted a diary study with ten programmers to closely examine the kinds of resources they refer to for different queries and how they navigate to that resource. Programmers search for information for many different reasons [77, 268], but we focused our study on programmers who were in the process of learning a new technology.

Goal

The goal of this phase of the research was to understand how programmers make decisions when navigating to a relevant resource.

Research Question

How can we represent the behaviour and rationale of programmers as they search for online resources when learning a new technology?

Publication

The study on understanding how people search for software documentation online was published in the article *How Programmers Find Online Learning Resources* [16].

Replication Package

The coded data set as well as the documents needed to replicate the study are available in our online replication package: <https://doi.org/10.5281/zenodo.7504510> [17].

3.1 Study Design

We conducted a diary and observational study we conducted, in which we closely examined how programmers find learning resources online. Diary studies provide a balance between observational studies in natural settings, observational studies in a lab environment, and surveys [136]. Our data collection approach is modeled after previous self-reporting studies, such as those conducted by Gallardo-Valencia and Sim [77] and Xie and Joo [270], that are followed by a reflection interview in which participants may be asked to recreate some of their searches during the study period.

We focus on programmers learning a new technology. We use the term *programmer* to denote people who write code in any capacity. This population includes *developers*, who are professionally employed to build and maintain software. We advertised our study on university email lists, public software technology email groups, as well as social media channels, and required interested participants to email the author of this thesis. While recruiting participants for the study, we ensured that they had prior programming experience, and were just beginning to learn a technology new to them. We ensured that no participants were learning only from in-person or online courses, pre-defined training material, or research papers, where their searches would be guided by instructors or training material that contained pre-defined learning objectives. We also enlisted only those participants whose learning happened on a regular basis, i.e. at least daily for a minimum of three days a week, so that the searches performed during the study would be part of a regular learning process, instead of an exceptional occurrence.

3.1.1 Data Collection

Each participant filled a form with demographic questions (see Appendix A.1). We asked each participant to fill in a diary entry for every search for information made online over a period of five days, regarding the technology they were learning. We requested that participants document every step in their search process. Figure 3.1 shows the diary template we provided. Participants were requested to send their completed diary entry (or entries) to us at the end of each study day. Thereby, we were able to immediately clarify with the participants any ambiguity or request for more details in the diary entries, if necessary.

After each participant completed five study days, the author of this thesis conducted an hour-long open-ended interview with the participant. The interviewer asked the participant to recreate two selected diary entries from the study week. While repeating the steps in the entries, the interviewer encouraged the participant to describe their thought process aloud [108]. We did this for two reasons. First, the interviewer could verify the accuracy of the diary entries, clarify any ambiguities, and correct any mistakes in reporting during the

Name:
Date:
What I am hoping to learn with my search:
I have searched for this information before: Y/N
Steps taken:
1. <Insert Step 1> Thought that guided this step:
2. <Insert Step 2> Thought that guided this step:
3. <Insert Step 3> Thought that guided this step:
[Please add more steps here if necessary]
I found the information I needed: Y/N
Approx. amount of time taken to find information: <X>
Comments and Notes:

Figure 3.1: Diary entry template for each search session.

interview. Second, we could gather rich descriptions of the search sessions from the participants, which were not present in the diary entries. Based on the course of the discussion, the interviewer asked follow-up questions regarding the participant's information seeking process. After the interview, we asked the participant to complete a questionnaire about their experience of looking for learning resource online and participating in the study (see Appendix A.2). We offered a compensation of up to \$100 CAD for completing the study. The study is approved by the Research Ethics Board Office at McGill University.

Eleven participants took part in the study, providing a total of 131 diary entries. One participant completed only two of the five study days, and submitted two diary entries. Since we received less than three diary entries (on average, one per day for the minimum criteria of learning three days a week) from this participant, we omitted their data in our study. Table 3.1 describes our participants' demographics.

Of the remaining 129 diary entries, we filtered out 14 entries from our data set because they were not searches for technical information about the technology (e.g. one entry was about industry perspectives of the technology), contained insufficient information about the steps to reproduce entirely, or accessed only resources beyond the scope of the study (such

CHAPTER 3. HOW PROGRAMMERS FIND SOFTWARE DOCUMENTATION

Table 3.1: Participant demographics for the diary study.

ID	Occupation	Prog. Exp. (Years)	Target Technology	Learning Phase
P2	Software Engineer	14	Elasticsearch-Logstash-Kibana (ELK)	Initial
P3	Research Assistant	6	Data visualization in Python	Intermediate
P4	Master’s Student	4	Genetic Algorithms for Automatic Software Repair	Initial
P5	Research Assistant	6	Object Oriented Programming in C++	Intermediate
P6	Software Engineer	6	Selenium using Python	Initial
P7	Undergraduate Student	2	C#/Unity	Initial
P8	Master’s Student	9	Torch + Lua	Intermediate
P9	Software Engineer & Master’s Student	10	DRM and VA-API	Initial
P10	Software Developer Trainee	5	SQL	Initial
P11	Software Developer	6	Microservice Mesh with Envoy and Istio	Initial

Prog. Exp. — Programming Experience

Learning Phase — “Initial” learning phase indicates 0-4 weeks of learning so far, “Intermediate” is any time beyond 4 weeks.

as research papers). Despite filtering these 14 entries, no participant had less than three valid diary entries. Our final data set comprises of 115 diary entries from ten participants.

Our participant sample size follows that of prior diary and observational studies including work done by Teevan et al. (15 participants) [241], Meng et al. (11 participants) [166], and Chattopadhyay et al. (10 participants) [46]. We discuss qualitatively observations based on the participants’ detailed experiences during this study. The sample size also ensured the study could be completed in a realistic time period. Each participant’s study period is five working days during which time we kept in touch with the participants, answering questions they had about the study, reviewing and requesting for clarifications in diary entries when necessary, and also performing iterations of coding of the entries. Analysing the transcripts of the hour-long interviews for relevant insights and useful anecdotes took an entire day each. Still, the 115 diary entries we collected from ten participants allowed us to make numerous repeated observations of components and connections between them (see Section 3.5).

3.1.2 Data Analysis

We refined the data collection method as observations emerged in the analysis [135]. This way, we were able to clarify ambiguities and ensure that the diary entries and interviews of participants remained within the context of our study.

Figure 3.2 illustrates the process we followed to obtain our data set. The author of this thesis open coded 25 diary entries from five participants, chosen in a stratified manner such that the sample consisted of one entry from each day from each participant. The open coding process included annotating the diary entry’s content for the information that the participant needed (QUESTION), and their thought process during the search session. We additionally identified the web links to resources in the diary entries. As a result, we created 132 codes in

CHAPTER 3. HOW PROGRAMMERS FIND SOFTWARE DOCUMENTATION

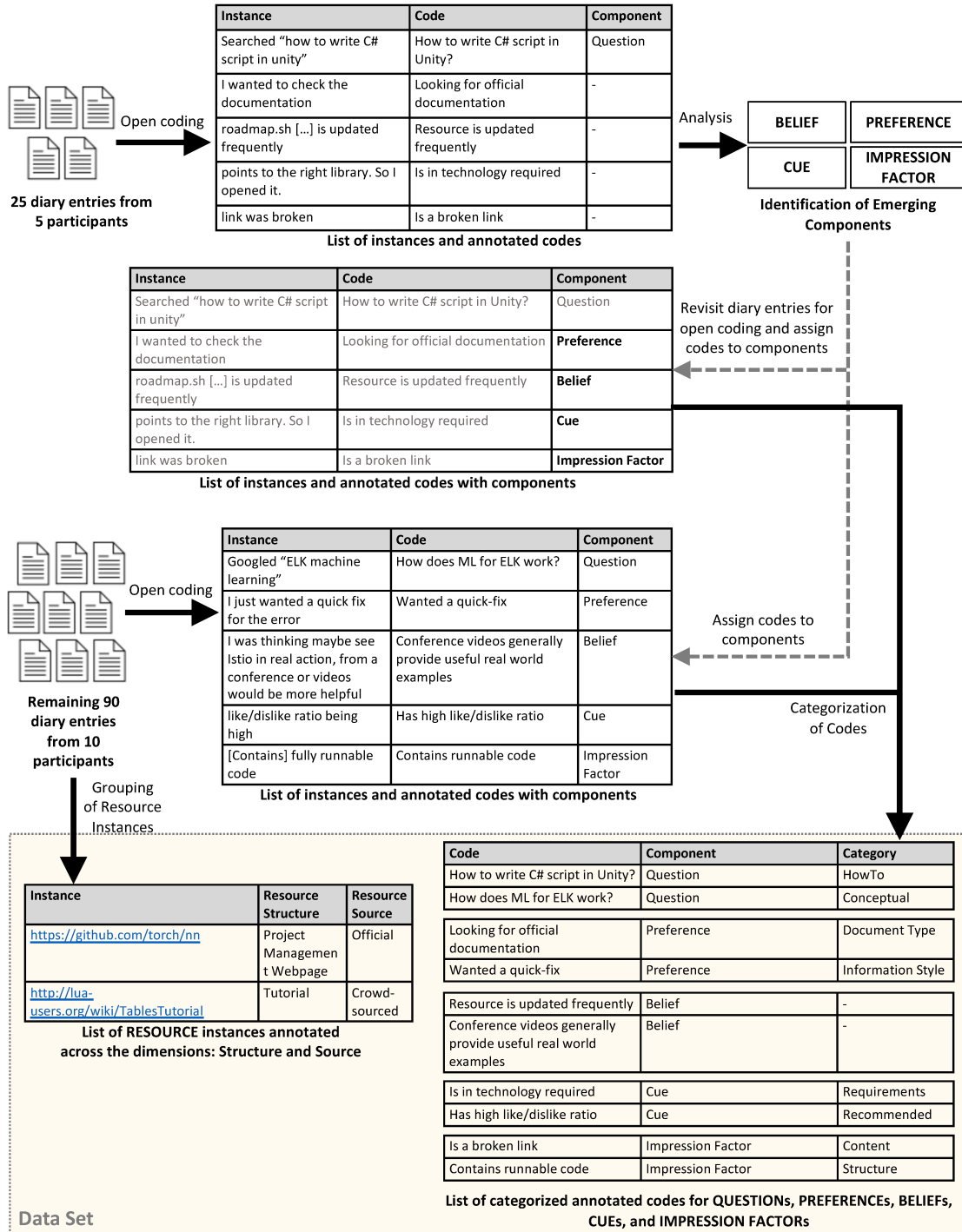


Figure 3.2: Process followed to obtain the data set for this study. The tables obtained in the last steps (represented by the last row in the figure), together form our data set.

total. Upon analysing the created codes, we identified four emerging components, namely, PREFERENCES, BELIEFS, CUES, and IMPRESSION FACTORS (see Section 3.2). The author then revisited the 25 diary entries to verify the open coding, identify missed instances of the components, and assign created codes to one of the five components QUESTION, PREFERENCES, BELIEFS, CUES, or IMPRESSION FACTORS. As the study progressed with new participants, the author open coded the new diary entries and associated these codes to the appropriate component.

To characterize each component, the author of this thesis performed card-sorting of all the codes within each of QUESTION, PREFERENCE, CUE, and IMPRESSION FACTOR. They created a coding guide for the different categories that they observed. To alleviate the bias of a single annotator, the authors' supervisors (also the co-authors of the corresponding publication) used this coding guide to categorize the codes, and we performed inter-rater reliability tests on their categorization. The Cohen's Kappa between the two latter authors for each component was 0.91 for QUESTIONS, 0.74 for PREFERENCES, 0.71 for CUES, and 0.79 for IMPRESSION FACTORS indicating at least substantial agreement in all four cases [131]. All three authors resolved the disagreements via a collective discussion.

For RESOURCES, the three authors together discussed and grouped the *instances* based on two dimensions: *structure* and *source*. We use the term *instance*, in the remainder of the article, to refer to each individual quotation in the diary entries that have been coded. It is possible that multiple instances have the same code if they are nearly identical in their semantics. We did not categorize BELIEFS further because they are described rarely by our participants.

We noticed that in the diary entries, there existed meaningful connections between instances of different components. For example, multiple RESOURCES were accessed to answer a single QUESTION, and different CUES were used to select each of the RESOURCES. Hence, for each search session, we identified these connections. We named the relation between two components according to the semantics of their connections.

We used statistical tests to investigate the association between different components in our data set, for example whether the *Authoritative* CUE was predominantly used to access *Official* documentation. We tested the null hypothesis that there is no association between any two categories of components [233]. We discuss the details of the statistical analysis in Section 3.5.

Threats to Validity

As part of our study, we required participants to self-report the steps they take in their search. This poses a threat to internal validity because the steps may not be reported exactly as they are performed. To verify the accuracy of reported steps, we performed an interview at the end of the diary study in which we asked the participants to recreate two search sessions. This way, we were able to determine that the reported entries are correct. In only three entries participants made corrections to their diary entries, and in no case did the corrections significantly impact our findings.

The analysis methodology involves manual annotation which are subjective. To alleviate

annotator bias and measure the subjectivity of the task, three annotators were involved in categorizing our data set. We measured the agreement scores between them and found that they had substantial agreement (see Section 3.1.2). To validate our coding guides, we asked two external annotators who had little to no context about our study to annotate our data set using the coding guides. They achieved agreement scores of 0.73 for QUESTIONS, 0.74 for PREFERENCES, 0.73 for CUES, and 0.63 for IMPRESSION FACTORS, indicating substantial agreement [131] for all four components.

We also face the threat to external validity, i.e. the generalizability of our observations to other programmers who are learning a new technology. Our sampling of ten participants does not allow a generalization from sample to population. This is inevitable for diary studies, and our sample size is consistent with the norm for this research method. The implication is that our observations may be limited to the behaviour of our participants. However, our goal is not to make claims about general population behavior, but to theorize the factors programmers think about as they access resources. Our findings are also supported by robust statistics of the relations between components in our data set. Thus, we refrain from making assumptions and comments about general behaviour of programmers. Furthermore, our proposed model represents the possible aspects of a search session. The current set of components may not be exhaustive, and can be augmented with additional components that may be observed in future work.

3.2 Resource-Seeking Model

We propose a model for representing how programmers seek online resources when learning a new technology. This model consists of six components: three need-oriented components (QUESTIONS, PREFERENCES, and BELIEFS), and three resource-oriented components (RESOURCES, CUES, and IMPRESSION FACTORS). All the components, except for BELIEFS, occur within the scope of a single *search session*. A search session is a time window in which a programmer searches for and navigates through one or more resources online to meet their information needs. We illustrate the components and relations in the resource-seeking model in Figure 3.3 using excerpts from our data set. We provide more details about each of the components in Sections 3.3 and Section 3.4 and the relations between them in Section 3.5.

3.2.1 Need-oriented Components

Since searching for a resource only arises when a programmer has some information they need to find, every search session *must* contain at least one instance of a QUESTION. A QUESTION refers to the search query that the programmer uses, and acts as the starting point as they begin their search.

PREFERENCES refer to a programmer’s pre-existing expectations or requirements of the resource they are looking for. In their search, a programmer may specifically look for an article on the blogging website *Medium*. A PREFERENCE *expands* a QUESTION by providing more context for the search.

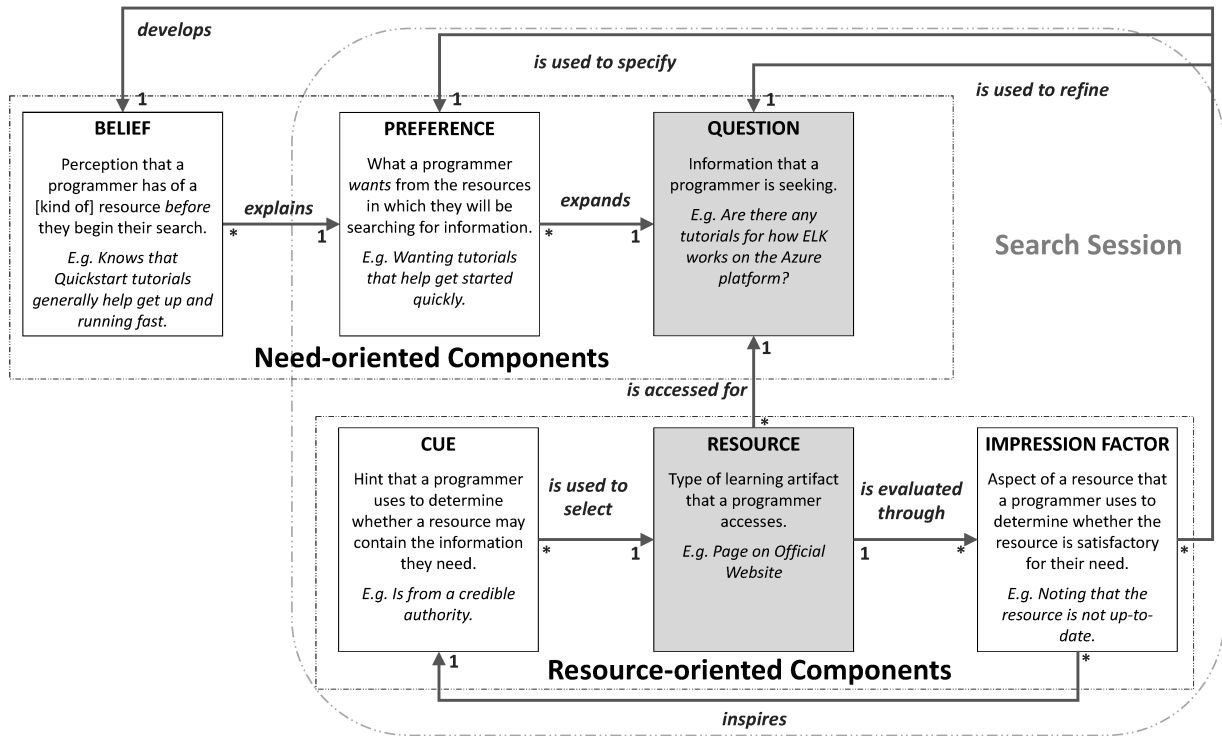


Figure 3.3: The online software technology resource-seeking model of users learning a new technology. The components shaded in grey, i.e. QUESTIONS and RESOURCES, are posited components, while the rest emerged from our analysis. The numbers and asterisk annotated on the arrows indicate the cardinality of the relation. For example, multiple CUES can be *used to select* one RESOURCE, and multiple RESOURCES can be *accessed for* a single QUESTION.

CHAPTER 3. HOW PROGRAMMERS FIND SOFTWARE DOCUMENTATION

Table 3.2: QUESTION categories.

Category	Description	Example	#
HowTo	Questions that ask how to achieve a particular functionality. This also includes how to navigate the development environment, for e.g. to build an application. This does not include how to fix errors, which should be categorized as <i>Debug</i> .	<i>How to get the dot product between two word tensors using nn.dotproduct?</i>	55
Conceptual	Questions related to the conceptual understanding of technology and its components. This includes whether a component exists, what a component is, its syntax, what the difference between some components are, and how multiple technologies interact.	<i>What is “shallow copy”?</i>	53
Document Type	Questions related to finding a useful resource for learning or determining what information a particular resource or kind of resource provides. The question includes looking for a particular resource, or a kind of resource. It also includes questions about what information a particular resource provides about the technology.	<i>What is a tutorial I can use to learn about plotly?</i>	25
Debug	Questions related to why an error occurs or how to fix it. This includes trying to understand what an error message means.	<i>Why does the stale element reference error occur?</i>	14
Misc.	Questions that can not be exclusively grouped into one of the other categories, or does not contain enough context to identify the appropriate category.	<i>What are some best practices when coding in Python in the Selenium framework?</i>	4

— The number of instances of each category in our data set.

A BELIEF is a pre-existing opinion about certain resource or type of resource. It *explains* a PREFERENCE, as it justifies the reason the preference arises. A programmer may explain that they want to watch a video because “[...] a Youtube video [is] easier to follow than a textual post that might contain more jargon I don’t follow.”

3.2.2 Resource-oriented Components

Within a search session, programmers access at least one RESOURCE to find the information they need, making it the other essential component in addition to QUESTIONS. A RESOURCE refers to a learning resource that is accessed by a programmer to find the information they need. A programmer may visit multiple RESOURCES within a single search session. We define a typed relation between instances of these two essential components: a RESOURCE *is accessed for* a QUESTION. For example, a programmer may search for “What does the valgrind error summary

mean?” To answer this question, a programmer may click on a video (the RESOURCE) from a search results page.

When presented with links and/or previews of resources, programmers use CUES, i.e. hints or characteristics of the resources, to make decisions about whether to access them. For example, a programmer may click on a resource because “[...] it would be the best source, since it is from the original makers of the [target programming] language”. Hence, a particular CUE *is used to select* a RESOURCE.

The IMPRESSION FACTOR of a resource is the aspect of the accessed resource that a programmer uses to evaluate the resource. A RESOURCE *is evaluated through* the IMPRESSION FACTOR. For example, a programmer may use the date of the last update of a resource to assess whether it might be out of date. The IMPRESSION FACTOR also plays an important role in the feedback loop for the search process. It *is used to refine* the QUESTION or *is used to specify* a new PREFERENCE as part of query refinement. For example, a programmer, upon realizing their query is returning only scientific papers, may choose to add “Medium” to the search query, and subsequently look only for resources hosted on the Medium website.

An IMPRESSION FACTOR can also be used to *inspire* a new CUE when searching for more resources. After finding a RESOURCE that “skipped a lot of basic information”, a programmer may be inspired to click the next RESOURCE if it from a website that hosts “entry-level tutorials for technologies”. An IMPRESSION FACTOR may also help *develop* a BELIEF that influences subsequent search sessions, if a programmer forms a strong impression of a RESOURCE.

3.3 Need-Oriented Components

We consider QUESTIONS, PREFERENCES, and BELIEFS as need-oriented because they involve aspects related to what the programmer is searching for.

3.3.1 QUESTIONS

We organized QUESTIONS into four categories (see Table 3.2). The most frequent (55 of 151) instances we observed are task-oriented **HowTos**, while the least frequent (4 of 151) instances are **Debug** questions. Both these categories have been identified in prior work by Rao et al. [204] and Gallardo-Valencia and Sim [77] as types of information need.

The **Conceptual** QUESTION takes four different forms. Participants asked *what is* questions when they wanted to understand the fundamental knowledge about a particular component, e.g. “What are graph objects (component) in plotly (library)?”[P3]. Some participants also wanted to understand how a certain concept could be *applied* in a concrete context. For example, P4 searched for “How is genetic programming applied to automatic software repair?”. Four of the participants had questions about the difference between two components or technologies, such as “What is the difference between softmax and softmin functions?”[P8]. P6 had four *syntax* related questions like “What is the syntax for do-while loop in Python?”.

The **Document Type** category refers to a search intended to find a particular resource. In some cases, the participants knew exactly the website or resource that they were looking

CHAPTER 3. HOW PROGRAMMERS FIND SOFTWARE DOCUMENTATION

Table 3.3: PREFERENCE categories.

Category	Description	Example	#
Resource Type	Specifying the resource or kind of resource needed.	<i>Looking for a Github resource</i>	27
Information Style	Specifying that the information should be structured or presented in a particular manner. This includes organization, level of granularity, depth, recency of information, and if code or data examples are wanted.	<i>Wants a resource that provides a high-level overview</i>	25

— The number of instances of each category in our data set.

for: “What information about Genetic programming for automatic software repair can be found on blogging website Medium?”[P4]. In other cases, the participants only had an idea of the *kind of* resource they were looking for. For example, “What is a tutorial I can use to learn about plotly?”[P3]. We observed that the *Document Type* searches occurred in two scenarios. Participants began a search session with a general query for useful resources. When searching for “What are some tutorials on the Elastic stack?”, P2 mentioned they “wanted to see what’s out there”. In the second case, participants specified the type of resource they wanted in the middle of a search session, normally after an unsuccessful search for information. They recounted past experiences and narrowed their search within familiar resources. For example, when searching “Genetic programming Automatic Software Repair” in the Youtube search box, P4 mentioned “After trying my luck with Google search, I wanted to see if there were any resources on Youtube”.

3.3.2 PREFERENCES

We elicit two categories of PREFERENCES (see Table 3.3). *Resource Type* indicates the specification of a particular resource or kind of resource. P2 specified in one search session that they “wanted in-depth API documentation”[P2], as opposed to resources that market the visualizations created using the technology. *Information Style* refers to the specification of the characteristics of the information. For example, P2 said in another entry: “I found a lot of useful search results ranging from specific API fields and tactics to more high-level overviews. Since I’m still learning, I went for the higher level overviews.”. Both categories of PREFERENCES are nearly equally frequent in our dataset (27 and 25, respectively).

The PREFERENCE plays an important role in the searching process because it describes the bias that the programmer has when looking for information. For example, a participant looking for a particular resource type may have entirely ignored other resource types, despite them containing the information they needed: “I found two results that looked promising as they were both related to torch and looked like documentation rather than Q&A by public. The remaining ones were on Stack Overflow which I ignored since I want to rely on the doc.”[P8]

Together, QUESTIONS and PREFERENCES constitute the complete picture of what a participant is looking for, thus indicating the information need.

3.3.3 BELIEFS

A BELIEF justifies the existence of a certain PREFERENCE, by explaining *why* the PREFERENCE exists. P4 explained their preference of a video as “I find it easier to follow along; I feel like I can process information faster and I also have the option sometimes to just speed up the video faster if it’s going slowly.” In another search session, P4 was specifically looking for blog articles on the website Medium *because* “I find it a good place for quick reads that aren’t very deep into the area”. Thus, BELIEFS do not guarantee that preferences will always remain the same, because participants may change their preferences between or within a search session.

We recorded 14 instances of BELIEFS in our data set. In all cases, the BELIEF is based on prior experience of the resource, and/or a general notion of what that resource would provide. For example, P2 searched for ‘roadmap.sh’: “There’s a really good resource I already know about called roadmap.sh. It’s updated frequently.”[P2].

3.4 Resource-Oriented Components

RESOURCES, CUES, and IMPRESSION FACTORS are resource-oriented components because they are anchored to the resources that programmers access during their search.

3.4.1 RESOURCES

Table 3.4 shows the types of resources that we observed in our study. Each instance of a resource access is characterized across two distinct aspects, i.e. structure (seven categories), and source (three categories).

With respect to the structural aspect, we observed that 45 resources accessed are traditional types of software documentation like *Reference* documentation and *Tutorials*, yet the majority of instances are unconventional learning resources. A total of 51 instances of resource accesses were to *Forums*, 41 of which are Stack Overflow posts. The popularity of Stack Overflow can be attributed to two reasons. First, Stack Overflow is often placed prominently in the first page of the search results page when searching for documentation by Google [246], biasing users to click on this resource: “Whenever I search for any results in the domain software development, I tend to see more of [particular] resources, for example, Stackoverflow comes first.”[P10] Second, Stack Overflow acts as a hub for programmers: “As a practice, I always tend to open the first search result and most of the time it happens to be from Stack Overflow, and as we know, Stack Overflow is the go-to place for us [programmers].”[P6].

Thirty-four of the resource accesses were to *Indexes*, which are directory pages that contain links to other useful resources. Of these, there are four cases where the participants found the information that they needed within the search engine results page, usually because of some keyword present in the result snippet. “Simple problem, didn’t even click on a link. The minute I saw to_string I knew how to use it because I’ve used it a few times before.”[P5] In eleven cases, participants used the search results to identify that they were not going to find the information they needed, and consequently either refined or aborted their search.

Articles are the resources that do not follow the structure of a tutorial, a reference

CHAPTER 3. HOW PROGRAMMERS FIND SOFTWARE DOCUMENTATION

Table 3.4: RESOURCE categories.

Category	Description	Example	#
Structure-based			
Forum	A post on Stack Overflow or another discussion forum	stackoverflow.com/questions/9695329/c-how-to-round-a-double-to-an-int	51
Tutorial	An instructive document that generally provides steps to follow to achieve a particular task	medium.com/@deependra.ariyadewa/envoy-in-kubernetes-373d5621e243	37
Indexes	A directory, registry or set of search results	virusu.github.io/3D_kibana_charts_vis/	35
Article	A Wikipedia entry, arbitrary article, or textbook page.	en.wikipedia.org/wiki/Genetic_improvement_(computer_science)	33
Project Management Webpage	A page in a project management (e.g. Github) repository, including an issue discussion thread	github.com/dzharii/awesome-elasticsearch	29
Video	A video (typically found via Youtube search)	youtu.be/6P1ivCvofuk	11
Reference	API reference documentation	intel.github.io/libva/group__api__core.html	8
Misc.	A resource that can not clearly be differentiated as one of the above categories	www.cs.swarthmore.edu/~kwebb/cs31/s14/stackframe.pdf	11
Source-based			
Official	Resource hosted on a technology's official website or by the company that created or is managing the technology	logz.io/blog/elk-stack-raspberry-pi/	82
Third-party	Resource is created or hosted by a single party that is not the creator	www.geeksforgeeks.org/this-pointer-in-c/	65
Crowd-Sourced	Resource contains content that is crowd-sourced	lua-users.org/wiki/TablesTutorial	52
Misc.	The source of the resource is unknown or multiple (in case of search results)	<i>Search Results</i>	16

— The number of instances of each category in our data set.

documentation or a discussion forum, e.g. a Wikipedia or textbook page. We found 33 instances of *Article* accesses in our study.

P8 made the most accesses to the *Project Management Webpage* type of *RESOURCES* with 15 accesses to Github repository pages. This was because the technology that the participant was learning, *torch*, as well as its documentation, are both hosted on the project management platform. Despite making only five Github accesses, most of P9’s searches were spent on Github. In this case, the technology’s documentation was mainly accessible via source code comments. As P9 described, “it was discouraging to see that upstream calls auto-generated documentation from the source their ‘official documentation’, [...] the actual source comments in the files for which documentation is *not* auto-generated seem to have quite a bit of information.”[P9] We leave the study of source code comments to future work because it is stored in resources that are not primarily in a human speech language. We also observed participants accessing *Videos*: “It [watching videos] is in general in my learning process. Even if it’s learning a new programming language sometimes, I like watching a brief video.”[P4]

In the source-based categorization, the majority (81) of resources are from *Official* sources, i.e. from the developers of the technology themselves or associated companies. Accesses to *Third-party* (65) and *Crowd-sourced* (52) documentation are nearly the same in our data set.

3.4.2 CUES

We elicit five types of *CUES* in our data set (see Table 3.5). The most frequent type with 66 instances is *Recommended* which indicates some implicit or explicit endorsement by other users, resources, or the search engine that this resource is useful. P7 explained “I clicked on this [first link] first because it was recommended by Google, so they had a little box with an excerpt of the information. This made me think the link would likely have the correct information”. P6 echoes this by simply stating “I have realized that the first result tends to meet my expectations”. We also observed that some participants opened multiple resources in different tabs and briefly look for cues to determine which resource to access. For example, the number of upvotes a resource has is usually not displayed in the search results, but within the resource page. P4 stated “I find the high like/dislike ratio a good indicator of the video being good and giving information correctly”.

The category *Requirements* indicates that the resource seems to fulfil the participant’s criteria. For example, after entering a query looking specifically for Medium blog articles, P4 “clicked on the second link that appeared, as I saw it was from medium.com”. *Familiarity* with a resource is also a *CUE* that the participants used. In most cases, the participants recalled accessing the resource or similar resources and used their prior experience to determine if the resource could be useful. For example, “I clicked on this because I have watched videos by this creator before and liked his teaching style.”[P7] In two cases, P8 clicked on the resource simply because they had clicked on it before, even though they did not remember their previous experience. P8 said “The second result, I opened anyway since I saw that I had visited this page before and I was curious to know if there was something interesting there.”

Participants clicked on resources because they came from an *Authoritative* source. P2 explained: “Since it is recommended by the elastic devs themselves, it is hard to go wrong.” The

CHAPTER 3. HOW PROGRAMMERS FIND SOFTWARE DOCUMENTATION

Table 3.5: CUE categories.

Category	Description	Example	#
Recommended	The resource is chosen because it is among one of the top four search results, is featured by the search engine, has high number of upvotes, claps, likes, etc., is explicitly recommended by users, mentioned in another resource, or is generally popular or well-known.	<i>Google expands the blurb so must be relevant</i>	66
Requirements	The resource is the exact or similar resource wanted, or contains the exact characteristics needed. For e.g., if the resource is up to date, is in the correct or related domain, is the correct level of granularity, or its information is presented, styled, or formatted in a manner that is needed or preferred. If it is mentioned that this resource is accessed <i>because</i> it was useful in the past, it should be categorized as Familiarity instead.	<i>[Resource] Provides a formal introduction with historic information (wanted)</i>	53
Familiarity	The participant has some familiarity with the resource or the content that it contains, generally prefers it, or has used the resource in the past and has had a positive impression.	<i>Is a Stack Overflow link (preferred for technical questions)</i>	44
Authoritative	The source of the resource seems to be a credible or reputed authority. This includes cases where the resource is from the developers of the technology themselves.	<i>Resource is a reputed training website</i>	35
Keywords	The search result title/snippet or resource contains keywords that were present in the query, or are relevant to their question.	<i>Resource title seems closest to the error</i>	31
Misc.	Cues that can not be exclusively grouped into one of the other categories, or does not contain enough context to identify the appropriate category.	<i>Similar to what the participant intended to search for next</i>	17

— The number of instances of each category in our data set.

comments from P3 and P8 echo the same point: “Since the url is plotly.com, it’s probably credible and good information”[P3], “I chose the official doc because I thought this would be more reliable.”[P8] However, reliability is not the only reason to choose a resource by an authoritative source. P2 explained “[...] I like to get started on the software page to make sure I see updates to API” indicating such resources are generally up-to-date.

The *Keyword* category describes when participants assessed words in the search results to determine whether it could be a useful resource. When searching for the difference between the softmax and softmin functions, P8 clicked on the third search result because it “looked more promising since the title and description contained both words softmax and softmin.”[P8] In one instance of searching how two technologies interact, P2 used the search term “kibana react visualization”. Of the search results, they said: “I would have expected an actual kibana-react app to show up higher in the search results...”. For P6, this expectation is so high that when clicking on the Stack Overflow post that is the first result, they said: “I do not usually read the question because I trust Google to give me the exact answer. I think it takes practice for us to get the right result as well from having to type the keywords.” Participants felt they need not consciously look for matching keywords in search results.

We noted cases where a participant used multiple cues to determine whether to follow a link: “I clicked on the second one because I hope that since it is listed second, it might be related [to the query] and also since it is from official Pytorch doc.”[P7] In this example, the participant considered that the resource is both *Recommended* and *Authoritative*. Sometimes, participants were unsure which resources may prove useful, because of the lack of clear CUES. P10 explained their strategy in one such case: “I open many links [from the search results page] in all new tabs. I do it for four or five links... I usually start from the first link.”[P10]

3.4.3 IMPRESSION FACTORS

We grouped IMPRESSION FACTORS into three categories based on what the participants used to form their impression about the resource (see Table 3.6).

In the majority (64 of 117) of cases, participants evaluated the *Content* of the resource, making comments about its quality such as whether the information was sufficiently detailed, beginner-friendly, or up-to-date. For example, P2 noted about a resource that “this was from 2017, so specs may have changed with more recent Raspberry Pis.” Participants also mentioned that resources were jargon-heavy or too advanced. P8 found that the Envoy official documentation “would require [readers to have] intermediate/advance knowledge of DevOps in order to quickly grasp the information”. The comments on content can be useful for resource creators to improve the information presented in it and make it more accessible to readers.

Many participants made comments about the *Pertinence* of the resource to their needs, including whether they found the answer to their specific question. P9 was looking for information about how a particular encoder works when they landed on a seemingly relevant project “ffvademo”. However, they said “I read the README on the repository and saw that ffvademo is actually a decoder, not an encoder”. Essentially, this type of IMPRESSION FACTOR is about the alignment with the programmers’ information need. When trying to find out how to determine whether a word embedding contains a particular word, P8 accessed a Stack Overflow

CHAPTER 3. HOW PROGRAMMERS FIND SOFTWARE DOCUMENTATION

Table 3.6: IMPRESSION FACTOR categories.

Category	Description	Example	#
Content	Comments related to the nature or quality of the information contained	<i>Information in resource is not friendly to Windows-users</i>	64
Pertinence	Comments that are about the specific context, such as a target domain, use-case or question	<i>Too early in learning [process] for this resource to be useful</i>	35
Structure	Comments related to the <i>organization</i> of the resource	<i>Contains good demo code</i>	18

— The number of instances of each category in our data set.

resource and commented about it: “The answers talk about how embedding layers work, which I am not interested in”.

Participants also assessed the *Structure* of a resource, noting the presentation of content it contained and the way it was organized: “This website was very helpful- it had different sub topics on the left and there were many examples that helped me understand the concepts well”[P10]. Structure-related comments can be useful for resource creators to reflect upon and improve the organization based design of the resource.

3.5 Relations Between Components

We identified nine unique relations among the six components in the resource-seeking model. Five of these relations are infrequent: *explains*, *inspires*, *is used to refine*, *is used to specify*, and *develops*. We discuss these relations qualitatively in Section 3.5.4. For the four frequent relations, i.e. *expands*, *is accessed for*, *is used to select*, and *is evaluated through*, we noted that some instances were more commonly occurring than others. For example, participants often referred to *Forums* to answer their *HowTo* QUESTIONS. To quantitatively analyze the coincidence of the relations we adopted the Fisher’s Exact Test.

Fisher’s Exact Test is performed on categorical variables where the frequencies of co-occurrence between categories may be below five, and was originally proposed for a 2x2 matrix. Because our contingency tables are larger than 2x2, i.e. the number of categories of some components are more than two, we approximated the p-value using 200000 Monte Carlo simulations [164].

Since three of the four relations to be statistically tested involve the RESOURCE component, we performed two tests per relations, i.e. one for each RESOURCE aspect: *Structure* and *Source*. Thus, we performed a total of seven statistical tests to determine if there is a significant association among the components in our model, one for each of the relations shown via dark grey arrows in Figure 3.4. To mitigate the Type-I error during multiple comparison

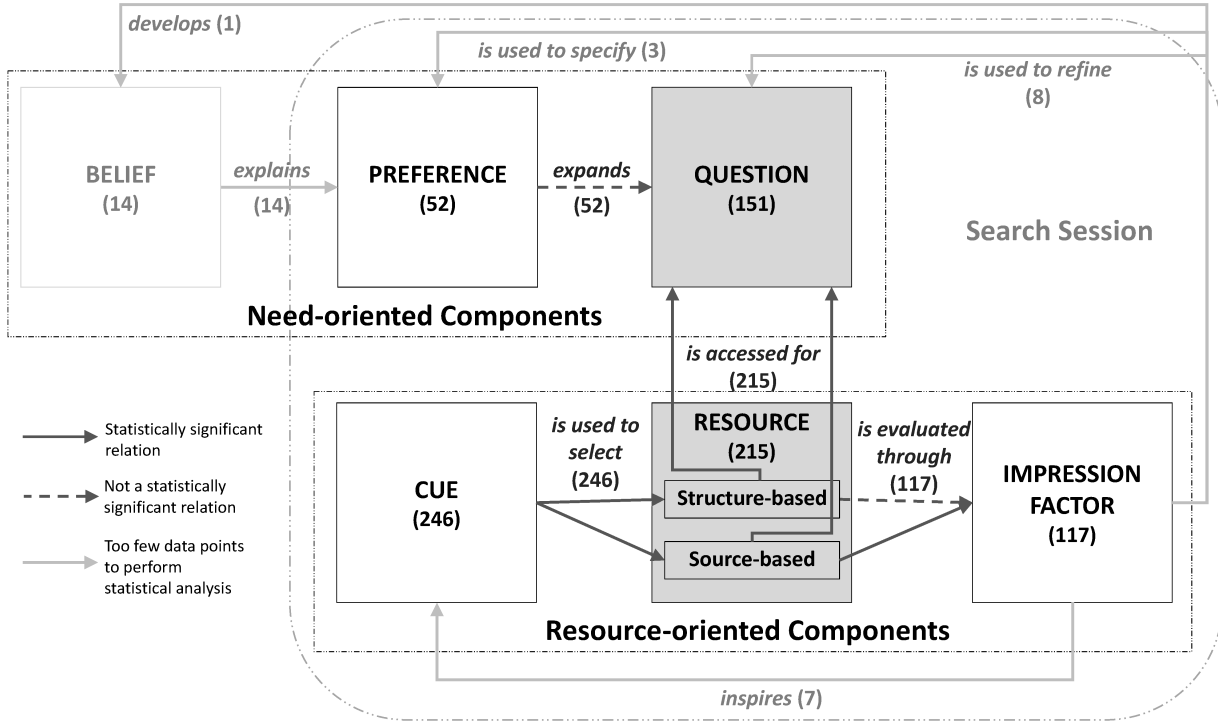


Figure 3.4: Frequencies of the components and relations of the resource seeking model, in our data set.

tests, we applied the Bonferroni correction to the p-values (i.e. multiplying each p-value by seven, one for each of the statistical tests performed)^(b) calculated via the Fisher’s Exact Test [3].

Table 3.7 shows the Bonferroni-corrected adjusted p-values (henceforth referred to as p-values) of this analysis. The two tests for the *is accessed for* relation between RESOURCE STRUCTURE and QUESTION, and RESOURCE SOURCE and QUESTION have a p-value lesser than α of 0.05. Similarly, the two tests for the *is used to select* relation between between RESOURCE STRUCTURE and CUE, and RESOURCE SOURCE and CUE, and the *is evaluated through* relation between RESOURCE SOURCE and IMPRESSIONS also result in a p-value lower than 0.05. Hence for these five tests, we reject the null hypothesis that the distribution of values between the component pair under test is due to chance, and we thus refer to the participating two components as “associated”. For the remaining two tests, i.e. *expands* relation between PREFERENCES and QUESTIONS and the *is evaluated through* relation between RESOURCE STRUCTURE and IMPRESSION, we cannot reject the null hypothesis. Figures 3.5 and 3.8 show the contingency table for these two relations.

For each identified pair of associated components, we computed the adjusted standardized residuals (henceforth referred to as *residuals*) for each cell in the contingency tables [224].

^(b)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 [49, 109, 265].

CHAPTER 3. HOW PROGRAMMERS FIND SOFTWARE DOCUMENTATION

Table 3.7: Bonferroni-adjusted p-values calculated by Fisher’s Exact test using 200000 Monte Carlo simulations of connections between each pair of model components.

	QUESTION	CUE	IMPRESSION FACTOR
Preference	1.0472	-	-
Resource Structure	3.5e-5	3.5e-5	0.4064
Resource Source	3.5e-5	3.5e-5	0.0022

The numbers in bold indicate statistically significant relations.

		Question					
		Document		HowTo	Debug	Misc	Total
		Conceptual	Type				
Preference	Type	7	13	5	2	0	27
	Style	12	5	6	1	1	25
	Total	19	18	11	3	1	52

Figure 3.5: Contingency table of *expands* relation between PREFERENCES and QUESTIONS.

Residuals are the normalized difference between the expected and observed frequencies of the relations, and reflect the effect size for the relation between two components in our model. Following common practice [224], if a standardized residual is greater than +2, we consider the effect to be meaningful and the associated relation to be “favored” by our participants. For example, we say that participants favored *Forums* to answer *HowTo* QUESTIONS based on the residuals, to indicate that the relationship “*Forums is accessed for HowTos*” occurs more than expected by chance in our data set. This does not necessarily mean that participants consciously expressed a favoritism for using *Forums* to answer *HowTos*.

When a residual is lesser than -2, we interpret the corresponding association as being “disfavored”. We examine the favored residuals (stated in bold) and contingency tables of associated components in more detail below.

3.5.1 RESOURCE *is accessed for* QUESTION

Figure 3.6 illustrates which RESOURCE *is accessed for* what kind of QUESTION.

Our analysis reveals that **participants favored *Articles for Conceptual* QUESTIONS. For *HowTos*, participants favored *Forums* and *Tutorials*. *Forums* provide flexibility to users to search for information within the exact context needed: “I feel I trust this (Stack Overflow) website, so I hope to find the answer [here]. If I don’t find the answer, there is an option for me to ask people for help. That gives me more leverage.”**[P10] Dondio and Shaheen showed that Stack Overflow can be as effective as a traditional textbook and course-based instruction for students to gain practical knowledge [59].

Participants also favored *Forums for Debugging* QUESTIONS. P8 said of their general search behavior: “If it’s a bug or an issue that’s not specifically working, I tend to go to Stack Overflow.”.

That **participants favored training resources like *Tutorials* to answer task-**

CHAPTER 3. HOW PROGRAMMERS FIND SOFTWARE DOCUMENTATION

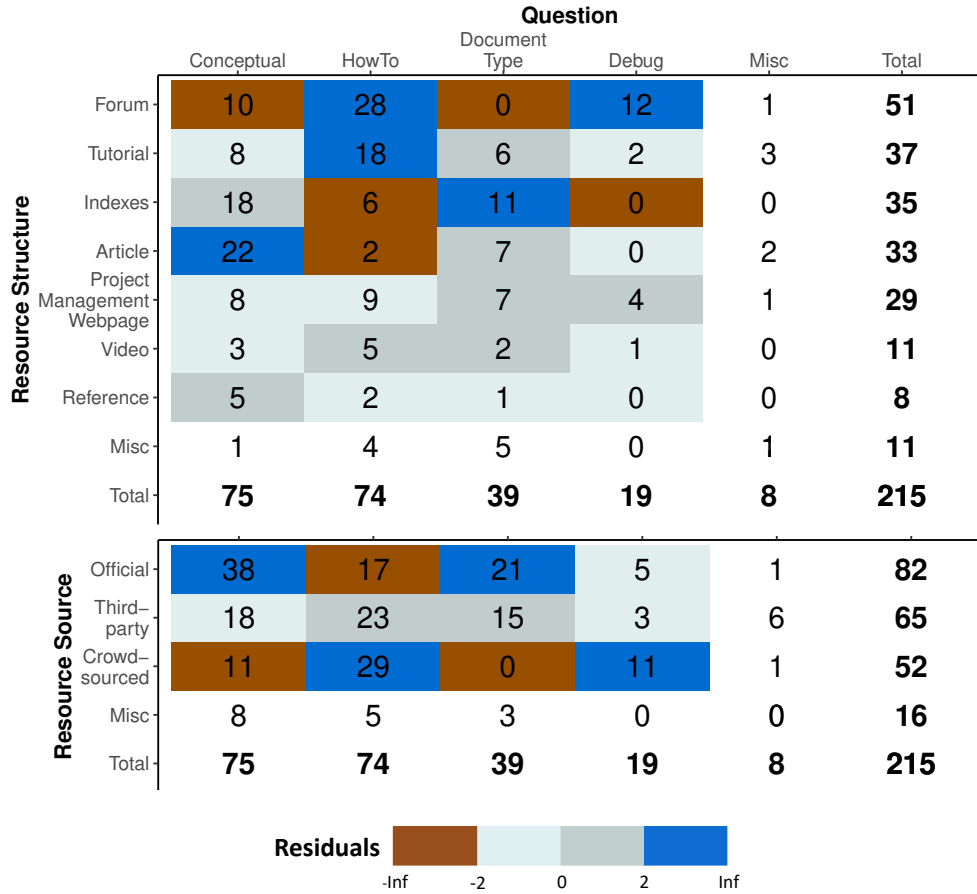


Figure 3.6: Adjusted Standardized Residuals and Contingency table for the *is accessed for* relation between RESOURCES and QUESTIONS. The values in each cell represent the frequency of connections between the pair of categories. Non-colored cells indicate that they were not included in the statistical analysis.

oriented *HowTo* questions, seems an intuitive result. Training materials have evolved to prioritize *procedural* information [43], i.e. information that directly supports actions, based on prior work that found software users use prior knowledge and procedures stated in text to perform tasks [153].

Participants favored *Indexes* for *Document Type* QUESTIONS. Ten of these eleven cases were to either home pages of a website (e.g. freedesktop.org) or directory pages of a particular topic on the website (e.g. www.elastic.co/demos). When participants used search results as a resource, it was useful to assess the pertinence of search results. For example, P2 looked at the search results and found that none were dated post-2017, and so they did not continue searching further. In another such case, P9 determined they would not easily find the resources containing the information they needed because “All the results looked auto generated (they had paths for page names)”, and so aborted their search soon after.

CHAPTER 3. HOW PROGRAMMERS FIND SOFTWARE DOCUMENTATION

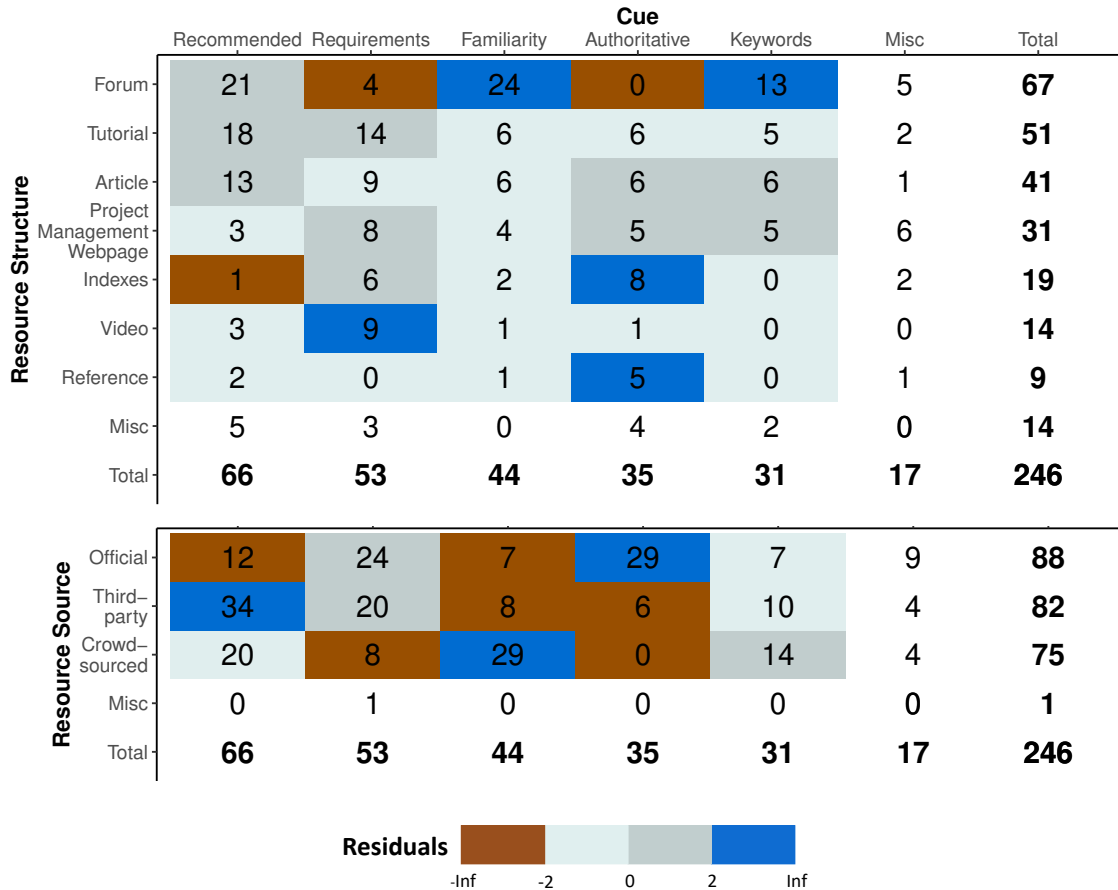


Figure 3.7: Adjusted Standardized Residuals and Contingency table for *is used to select* relation between RESOURCES and CUES. The values in each cell represent the frequency of connections between the pair of categories. Non-colored cells indicate that they were not included in the statistical analysis.

Across the source dimension, *Official* RESOURCES were favored by participants for *Conceptual* and *Document Type* QUESTIONS. Whereas, participants favored *Crowd-sourced* RESOURCES similarly to *Forums*.

3.5.2 CUE is used to select RESOURCE

The reason *why* a particular resource is accessed varies for different resources (see Figure 3.7). Participants favored the *Requirements* CUE when selecting *Videos*. This can be as broad as believing that watching a video would be better than reading text: “I figured I would find a youtube video on it more easy to follow than a textual post that might contain more jargon I don’t follow.”[P5], or as specific as that the video is short: “Clicked on this because its short length compared to others.”[P7].

Familiarity was favored when selecting *Forums* and similar *Crowd-sourced RESOURCES*. P10 points out: “It has come over time that I have the bias that these [W3Resource, Stack Overflow] websites are good, because I find many answers in them.”

Participants favored *Keywords* when accessing *Forums*: “I clicked on this first because it had the same wording as what I was looking for. So it made me think that it would be a good place to find the answer to my question.”[P7]

The *Authoritative CUE* was favored for selecting *Reference* documentation and *Indexes* by participants. The former link is intuitive since reference documentation generally accompanies the technology as *official* documentation. However the latter link is less obvious and is likely because when accessing a home page of a technology or documentation, participants consider the credibility of the source of the web sites they are clicking on.

Participants favored the *Recommended CUE* for *Third-party RESOURCES*. A majority of such connections occur because the RESOURCE was featured by the search engine. This shows that participants strongly trust a search engine’s ranking algorithm to suggest a pertinent RESOURCE.

3.5.3 RESOURCE *is evaluated through* IMPRESSION FACTOR

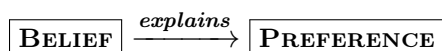
Figure 3.8 shows the contingency tables for the *is evaluated through* relation between RESOURCES and IMPRESSION FACTORS. While RESOURCE SOURCE is associated with IMPRESSION FACTOR, RESOURCE STRUCTURE is not.

Participants favored evaluating *Official RESOURCES* based on their *Content*. This may be because a certain level of quality is expected of a resource if it is from an authoritative source, especially if the original technology is well presented: “MITRE is a pretty detailed framework so I wasn’t surprised that their info [in the official documentation] was in-depth and dense”[P2].

Evaluating *Pertinence* to participants’ context was favored for *Crowd-sourced RESOURCES*. While in some cases, participants were able to find the information they needed, in others, they faced various issues while looking through *Crowd-sourced* resources. This included not finding posts that ask the same questions they have and not finding any answers to a post. Additionally, despite a lot of information available online, participants found it difficult to find the answer to their exact questions. P5 explained after unsuccessfully looking through three resources: “I felt like I was looking for an answer that was obvious, but I was only seeing questions that I wasn’t asking”.

3.5.4 Infrequent Relations

We observed an additional five relations that each occurred less than fifteen times. Except for the relation, BELIEF *explains* PREFERENCE, the other four relations involve a change in thought process or action based on the impression of a resource.



In fourteen cases, we observed participants describing the reason for their PREFERENCE based on an existing BELIEF. The BELIEF is usually developed from prior search experiences. In a

CHAPTER 3. HOW PROGRAMMERS FIND SOFTWARE DOCUMENTATION

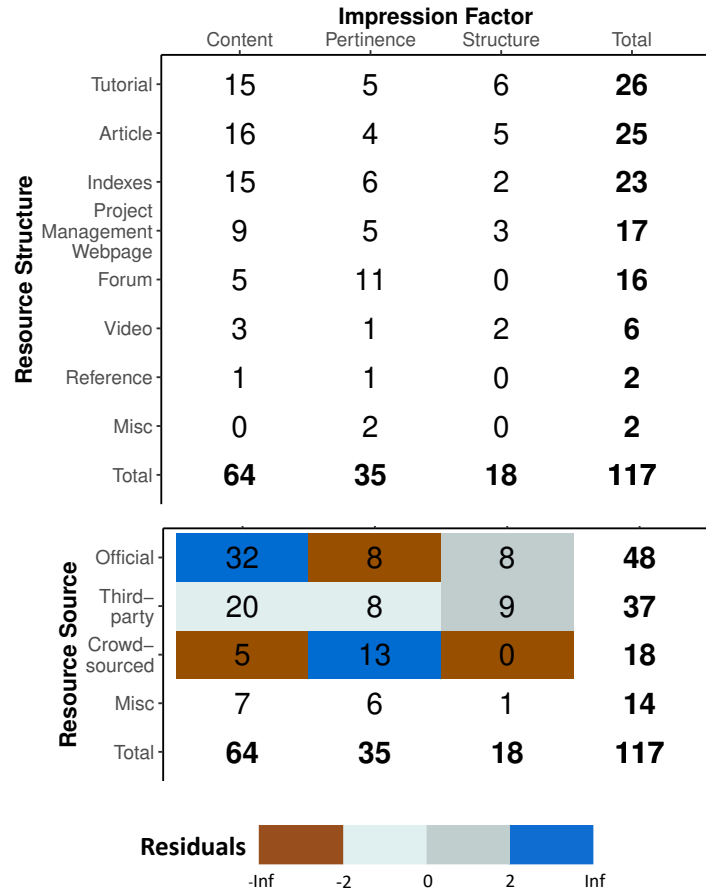


Figure 3.8: Contingency table of *is evaluated through* relation between RESOURCES and IMPRESSION FACTORS.

majority of cases, a BELIEF was used to explain a PREFERENCE on the *Resource Type*, either describing the specific website or a format. P4 explained their PREFERENCE for articles hosted on the website ‘Medium’: “I have found some very useful articles in Medium before, when learning about a certain area, and thought it would be a good resource again to learn something new. It usually has a lot of visual examples and explains things quite well.” In two cases, the BELIEF explained a PREFERENCE on the *Content Style*, once while looking for an “easy explanation” and the other when looking for “code examples”.

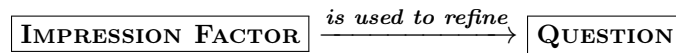


We observed seven instances in which an IMPRESSION FACTOR explicitly *inspired* a new CUE. When a participant initially browses through resources, they may use some CUE to assess the pertinence of a resource. After clicking on a resource and evaluating it, they may realize that there is an additional criterion they need, and employ it as a future CUE when searching

CHAPTER 3. HOW PROGRAMMERS FIND SOFTWARE DOCUMENTATION

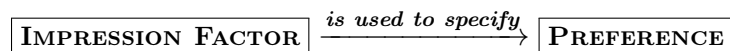
for more resources. This relation occurs between accesses of multiple resources for the same question within a single search session.

When searching for how to deploy Envoy in Kubernetes, P11 landed on the first tutorial in the search results. However, upon reading through it, they discovered that “the author skipped a lot of basic information ... because he assumes people who read it will have advanced knowledge [of] and familiarity with Kubernetes”. The participant proceeded to another resource, a blog hosted on ‘Medium’, stating that “Usually Medium contains a lot of entry-level tutorials for any technology”. Thus their new CUE of looking for a beginner tutorial was inspired by their impression of the previous resource. Two other instances of the relation involved finding a resource too advanced, and thus using a new CUE to find a resource that is better adapted to beginners.



In eight instances, participants refined their search query as a direct consequence of assessing a RESOURCE as not suitable to their needs. This relation occurs within a single search session but between two iterations of searching for similar information.

Most instances of this relation resulted from the unsatisfactory outcome using a first search query. In two cases, participants made the assessment on the first few search results: “None of the first links looked like it might provide an answer, so I searched again.”[P5] In these cases, the original query changed only by one or two terms. In other cases, participants realized the resources did not have the exact information they needed. After finding a Stack Overflow link with a question that did not exactly match the P5’s QUESTION about a particular line in a “valgrind” error summary, the participant realized their query may be too specific. They reframed their search query to look for general information about the error message, instead. Query refinement may also involve moving search platforms, as in one instance where the participant switched from searching on Google to on Youtube directly.



In addition to refining a question, in three instances, participants specified a new PREFERENCE based on their impression of a RESOURCE. In two of these cases, the motivation to refine the query was an incorrect *Resource type* of the resources. P4 scrolled through search results after searching for “Genetic programming Automatic Software Repair” but did not find a web page that was not a scientific paper. They refined their query, and additionally stated the specific resource they would be looking for going forward: “As I didn’t want to read a scientific study as an introduction, I tried my luck on finding it on Medium, as I find it a good place to have quick reads that aren’t very deep into the area.”[P4]

In the third instance of this relation, a participant introduced a new *Content Style* PREFERENCE, after reading a Stack Overflow post that did not provide a “precise understanding”[P5] of “shallow copying”. P5 subsequently stated that they were looking for “a nice and easy explanation of a concept”.



An IMPRESSION FACTOR can also play an integral role in influencing future searches for information. For example, a positive impression of a resource can result in a searcher returning to this resource during future searches. In one search session, P2 came across a guide hosted on <https://logz.io>, and found it to be an “exceptional” tutorial. They said, “the high quality of Logz.io search results, especially for this query, will probably make me look them up first for future learning in DevOps”[P5]. Although, we did not observe the conversion of the IMPRESSION FACTOR into a BELIEF within our limited study days (and thus the connection does not exist in our data set), P2’s statement suggests the potential for an IMPRESSION FACTOR to form the foundation for a new BELIEF.

3.6 Implications

Our resource-seeking model captures the different factors that play a role in the thought process of programmers as they navigate to a resource that could answer their question. It supports and complements prior work on finding pertinent information in software documentation (see Section 2.1). Our observations of the types of PREFERENCES, CUES and IMPRESSION FACTORS and how they relate to the QUESTIONS and RESOURCES involved in a search session provide insight to resource creators on how to improve the appeal of a resource for their target audience. Our results also have implications for the design of search tools and can help programmers improve their on-line search techniques: “I noticed some patterns that I usually follow and I thought about how I can improve them.”[P8] We discuss important observations and their implications from our study below.

PREFERENCES, potentially backed by pre-existing BELIEFS, are used to elaborate criteria for searching for resources to answer QUESTIONS. We observed that participants sometimes had expectations of the resources, prior to their search, and justified these expectations by their prior experiences. For example, P2 searched for “awesome ELK stack github”, explaining that “My experience with the awesome lists is that they’re both open source and up to date”. In our model, we formalize this behavior with the two relations: BELIEF *explains* PREFERENCE which *expands* QUESTION. *Equipped with the knowledge that programmers could have preferences during resource seeking, resource creators can study the behavior of target audiences to gain insight about their expectations, and thus make conscious decisions about whether to satisfy identified expectations and how to do so.* P2 and P11 mentioned that resources did not provide sufficient hands-on learning material such as practice questions, and their content was not well supported by diagrams, respectively. Upon identifying this preference of *Content Style*, resource creators can consider the trade-offs of adding these types of content in the resources to increase appeal [24]. *Search tools can be enhanced by allowing programmers to customize their needs based on their criteria.* P4 suggested a filtering mechanism that would allow programmers to specify the types of resources to search among, so that they would not have to wade through long results pages of non-preferred resource types. Such a filter would allow programmers to specify their *Document Type* PREFERENCE while searching.

Our participants' search behavior illustrates that **they accessed different RESOURCES to answer different types of QUESTIONS** when learning a new technology. All participants in our study accessed more than one type of resource through the study. Thus, searchers are forced to access multiple resources to satisfy their information needs, despite prior work discovering that there is some correspondence in information between different documentation types [15]. Furthermore, our analysis of the *is accessed for* behavior reveals that some resources are favored when answering particular types of questions. Whether this is because of the participants' implicit thinking process, or because of the nature of the resources themselves, requires further investigation. *Resource creators can be informed by this access behavior to tune particular types of resources to answer particular types of questions, allowing search tools to efficiently direct searchers to appropriate resources.* Meanwhile, *a centralized page indexing all the pertinent resources for learning a new technology would be useful for programmers to navigate the resource space.* Two participants mentioned they would prefer documentation to be standardized for ease of learning. P3 quoted the neatness of Java API documentation, explaining that other programming language documentation should follow suit. In the questionnaire, the other participant went as far as suggesting: "It would be nice if all software distributed complete man pages and info pages with documentation." [P9]

There exist visible and non-visible hints, or CUES, related to the quality and familiarity of a resource, to determine whether it is pertinent to information needs. Our model reifies the "scents" that information seekers use to find the information they need, according to information foraging theory [194], as CUES. We observed that these CUES can be either *objective* or *subjective* to programmers' wants and needs. For example, while the *Recommended*, *Authoritative*, and *Keywords* CUES are relatively objective, the *Requirements* and *Familiarity* CUE are influenced by the search context and the programmers' mindset. Prior work has focused largely on the *objective* CUES (see Section 2.1.3). However, our observation that participants use different CUES to select different RESOURCES suggests that *it is also important to incorporate search customization to support programmers in their use of subjective CUES.*

Different IMPRESSION FACTORS may be used to evaluate different RESOURCES. We observed that participants use different criteria to evaluate the quality and usefulness of a resource, especially depending upon the source of the resource. Particularly, some participants mentioned that learning resources should be easy to understand for beginners. P5 explained why they avoided a certain C++ *Forum*: "[...] the people who answer questions there get into a lot of detail using words that I don't follow." They explained that the case is different with Stack Overflow answers, where predominantly lesser technical jargon is used making answers easier to understand, thereby motivating their use of the web site. P11 also stated that searching is especially tougher for beginners as it is expected that they are aware of technical jargon - an interesting paradox since beginners are still in the learning phase of technical terms and details. *Our model provides the dimensions of a resource that programmers would use to evaluate it, and thus, the aspects of quality that resource creators must consider.*

A feedback loop can be formed using IMPRESSION FACTORS during the search process. We observed that participants used IMPRESSION FACTORS to refine their questions and clarify their criteria of resource type and content style. For example, while looking for gen-

eral information about what graph objects are in the plotly library, P3 found a resource that contained too much text. After accessing this resource, they explained that their PREFERENCE was resources with precise information, such as a list of methods and how to use them. Only after evaluating a RESOURCE, did they consider the PREFERENCE more seriously and use the criterion as a CUE for the subsequent resource access. *Thus, IMPRESSION FACTORS encourage programmers to reflect on their criteria and context during search. Furthermore, IMPRESSION FACTORS can be used to inform the creation of tools that assist in query refinement during the resource seeking process.* [149]

Chapter 4

Documentation Properties and Styles

We noted that programmers seeking information face a number of choices, and use *cues* to make decisions about resources to access (see Chapter 3) [16]. Some search engines incorporate cues related to *information content* to alleviate the amount of time and effort that programmers take in finding pertinent information. To assist programmers in their manual search process, prior work has explored faceted searching [116, 117], i.e. searching with content-related categories, and providing estimated time cost of reading search result web pages [112]. Still, to develop documentation that can cater to the needs and preferences of users, it is also important to understand how existing resources are designed.

We investigated how current online documentation, specifically programming tutorials, are designed. Despite many recommendations for how a tutorial should be structured, no universal standard is followed to design tutorials. Thus, tutorials about the same programming languages vary in their content and presentation. We complement prior work that focuses on information content of resources, with a framework to characterize resources based on their design properties and *styles*. Together with information retrieval techniques, our framework can be used to identify appropriately styled pertinent resources for different programming contexts.

Goal

The goal of this phase of the research was to determine how tutorials vary in their properties and investigate the ability to define a tutorial as being of a particular *style*.

Research Question

1. *To what extent do software tutorials vary in their structure and content-related properties?*
2. *How can we systematically reason about the design of software tutorials?*

Publication

The study on understanding the variations across popular programming tutorials was published in the article *Properties and Styles of Software Technology Tutorials* [19].

Replication Package

The details of our resource collection process, the extracted property data, and the results of the analysis are available in our online replication package: <https://doi.org/10.5281/zenodo.10048532> [18].

4.1 Data Collection

We focus on tutorials of the top five most popular technologies according to Github for 2021: Java, C#, Python, Javascript, and Typescript.^(c) We extracted the properties of web pages of tutorials for these technologies.

4.1.1 Resource Collection

We identified popular tutorials for each software technology through a manual online web search using the search engine DuckDuckGo.^(d) For each programming language, we collected the common non-advertisement search results on the first three pages for the three queries “<language> tutorial”, “<language> programming tutorial”, and “<language> development tutorial”. We used the *common* search results as a proxy for *popular* tutorials because they are consistently top-ranked by the search engine.^(e) To create a data set of comparable tutorials, we filtered out tutorials that did not fit the scope of our work, i.e. multi-page comprehensive tutorials.^(f) The websites on which the tutorials are hosted include technology websites (e.g. Oracle, Mozilla), those dedicated to tutorials for a single language (e.g. PythonTutorial, TypescriptTutorial) and those dedicated to tutorials for multiple technologies (e.g. BeginnersBook, Programiz). We retrieved traffic-related metrics for each website from [similarweb.com](https://www.similarweb.com) to investigate the popularity of each website hosting tutorials.

We analyzed each *resource*, i.e. an individual web page referenced by a given URL in a tutorial, because each page is indexed separately by a search engine. Thus, a tutorial *contains* multiple resources, and a *website* can host multiple tutorials. We discarded resources that had

^(c)<https://octoverse.github.com/#top-languages-over-the-years>

^(d)Despite video-based learners reported to have higher success rates than text-based learners [251], the majority of results from a standard search engine are textual documents [96]. Thus, we focused our study on text tutorials.

^(e)Although some websites host tutorials for languages within the scope of this study, e.g. TechBeamers offers a Java tutorial, we did not include it in our data set as it was not consistently top-ranked among our queries.

^(f)The exact steps of the resource identification and filtering process are available in the replication package and in Appendix B.1.

Table 4.1: Details about the programming language and host website of the resources studied.

	Domain	Java	C#	Python	Javascript	Typescript	Monthly visits (millions)	Pages/visit	Visit duration (mins)
BeginnersBook	beginnersbook.com	100	-	24	-	-	0.212	2.33	2.73
DotNetTutorials	dotnettutorials.com	-	105	-	-	-	0.847	2.37	3.13
Educba	educba.com	-	184	-	-	-	2.697	1.54	1.20
GeeksForGeeks	geeksforgeeks.org	-	30	-	-	-	0.028	2.21	0.36
Guru99	guru99.com	71	25	71	-	-	6.551	1.56	1.55
Info	javascript.info	-	-	-	92	-	1.841	2.52	3.05
JavaTPoint	javatpoint.com	79	121	73	196	45	16.840	2.08	3.97
LearnPython	learnpython.org	-	-	27	-	-	0.630	2.51	3.37
Mozilla	developer.mozilla.org	-	-	-	18	-	25.180	2.07	3.43
NetInformations	c-sharp.net-informations.com	-	96	-	-	-	0.057	1.80	1.52
Oracle	docs.oracle.com	328	-	-	-	-	9.899	3.31	3.35
Programiz	programiz.com	117	-	52	-	-	11.730	2.04	3.65
PythonDocs	docs.python.org	-	-	16	-	-	7.235	1.95	2.63
PythonTutorial	pythontutorial.net	-	-	180	-	-	0.596	1.88	3.80
SPGuides	spguides.com	-	-	-	-	6	0.248	1.34	1.48
TechBeamers	techbeamers.com	-	-	50	-	-	0.098	2.03	1.73
TutorialKart	tutorialkart.com	-	-	-	-	19	0.528	1.79	1.37
TutorialsPoint	tutorialspoint.com	39	-	28	36	21	20.620	1.78	2.68
TutorialsTeacher	tutorialsteacher.com	-	59	-	-	-	1.483	2.89	2.37
TypescriptTutorial	typescripttutorial.net	-	-	-	-	50	0.107	3.61	4.23
W3Schools	w3schools.com	54	35	44	-	-	57.620	3.71	6.31
W3SchoolsBlog	w3schools.blog	-	-	-	-	31	0.683	2.45	1.48
WebTrainingRoom	webtrainingroom.com	-	31	-	-	-	0.020	1.27	0.62
Total		788	686	563	342	172			

The last three columns are estimations of the website traffic from pro.similarweb.com for the period Apr.-Jun. 2023.

obfuscated HTML,^{(g)12} did not provide technical information about the target programming language, or followed a recognizable non-tutorial format (e.g. Q&A, exercises). We collected a total of 2551 resources hosted on 23 websites for five programming languages, as shown in Table 4.1.

4.1.2 Property Extraction

For each resource, we automatically extracted each top level HTML element within the manually identified main content element, and refer to these as *blocks* (see Figure 4.1).^(h) Table 4.2 describes the properties we extracted for each block in every resource, and the rationale behind identifying these properties. When calculating the length of a table in number of cells, we did not count cells that contain three or fewer characters, identifying them as index cells. For example, for the table in JavaTPoint’s “Module vs. Namespace” resource,³ we disregarded the first column, and identified the total size of the table as 22 cells (including headers).

To identify topics mentioned in resources, we used the JSI Wikifier,⁽ⁱ⁾ which identifies topics covered by Wikipedia articles in target text. Nassif and Robillard proposed a whitelisting technique for computing topics, with which the JSI Wikifier achieved up to a precision of 0.95 at the expense of lower recall, on a set of 500 Stack Overflow posts [175]. Since we focus

^(g)Links to resources are present in the section Resource References.

^(h)We used Python v3.9 and the library `beautifulsoup4` to parse HTML.

⁽ⁱ⁾<https://jsi-eubusinessgraph.github.io/jsi-wikifier-api/>

Table 4.2: Properties extracted at the block level for each resource.

Property	Description	Applicable to Block Type	Rationale
type	Whether the block is primarily a header, text, code, table, or image.	all	Identifying the type of content in a resource provides insight on the extent to which resources cater to the preferred visual stimuli as opposed to only text, in information resources [69].
contains_image	Boolean of whether the block contains an image within it.	text	Depending on the structure of the HTML, a text paragraph may contain an embedded image. We include such images, in addition to image-only blocks when calculating resource properties (see Table 4.3).
header_depth	Depth of the header. For example, the depth of an h3 block is three.*	header	Documentation writers must consider that readers interacting with technology may have a number of branching use cases for which they may consult particular parts of documentation [258]. The level of fragmentation, given by the depth of sections, provides insight into the extent of breadth versus depth of the content in the resource.
size	Size of the block in terms of number of sentences, lines of code, or number of table cells, as applicable.	text, code, table	The length of sentences [122] and length of code snippets [39] are fundamental aspects that are used to measure the readability of text and code respectively.
wiki topics	Topics covered that correspond to computing-related articles on Wikipedia.	text	The amount of topics to cover in a tutorial is a deliberate design decision that creators must consider [24].
task phrases	List of instructional-styled task phrases contained in the block.	text	Task phrases can help determine the information style of a resource as either procedural or declarative in nature [119].
links	The set of hyperlinks within a block, including internal links, i.e. referring to pages within the same tutorial, or external links, i.e. referring to pages outside the tutorial	text	Whether to delegate some information in a resource to other resources by provide references to other web pages has been discussed as a consideration for designing resources [24, 258].

* We account for relative header depths when computing the resource-level property *Maximum header depth* (see Table 4.3)

```

▼ <main class="content">
  ▼ <article aria-label="Constructors in Java – A complete
    study!!" class="post-1600 post type-post status-publish
    format-standard category-oops-concept entry">
      ::before
      ► <header class="entry-header">...</header>
      ▼ <div class="entry-content">
        ::before
        ► <p>...</p>
        ► <p>Constructor has same name as the class and looks
          like this in a java code.</p>
        ► <pre class="prettyprint prettyprinted" style>...</pre>
        ► <blockquote>...</blockquote>
        ► <h2>How does a constructor work</h2>
        ► <p>...</p>
        ► <pre class="prettyprint prettyprinted" style>...</pre>
        ► <p>...</p>
        ► <p>...</p>

```

Figure 4.1: Example of identified blocks for the BeginnersBook resource “Constructors in Java”. We manually identified that the `article` tag inside the `main` tag with class “content”, contains the main content of the page. We treated each of these elements, such as the `<p>` and `<pre>` elements boxed in red, as an individual block.

on identifying only relevant topics in a block, this trade-off is acceptable for the present study. The JSI wikifier is also easily accessible via its API and does not impose resource restrictions.

We used the TaskNavigator tool to extract task phrases [247], i.e. actionable instructions in software documents, e.g. “download package”. Treude et al. reported that their tool worked accurately to extract task phrases for 90% of 376 randomly selected sentences (from 17,448 sentences) in Django documentation. The identification of task phrases helps provide insight into whether a text may be *procedural* [119] in nature. We disregarded text blocks with fewer than 40 characters if they did not contain any *task phrases* or *topics* to reduce the effect of non-information phrases [152] such as “Output:”.⁴

We extracted block-level properties for a total of 98,915 blocks. We then aggregated the block-level properties to obtain resource-level properties. We normalized the aggregated block-level properties to account for dependencies of the properties on the length of the resource. The fifteen numeric resource-level properties (henceforth referred to as *properties*) that we computed are shown in Table 4.3.

4.1.3 Limitations

Our property extraction technique depends on the source HTML of the resources, causing it to be prone to error in the case of noisy, or inconsistent structure. For example, the PythonDocs Appendix resource⁵ does not follow the same HTML structure as other resources on the same website. Our automated property extraction retrieved zero text blocks, causing all text-related properties, i.e. task phrases, topics, and links to also be zero. We also leveraged external tools such as the JSI Wikifier and TaskNav to assist in the extraction of properties, despite them having imperfect accuracy and precision. We made deliberate decisions based on our investigation of the resources. For example, we did not count cells

Table 4.3: Computation of resource-level properties.

Property	Computation
Number of blocks	Total # of blocks
Proportion of text blocks	$\frac{\# \text{ of text blocks}}{\# \text{ of blocks}}$
Proportion of code blocks	$\frac{\# \text{ of code blocks}}{\# \text{ of blocks}}$
Proportion of header blocks	$\frac{\# \text{ of header blocks}}{\# \text{ of blocks}}$
Proportion of table blocks	$\frac{\# \text{ of table blocks}}{\# \text{ of blocks}}$
Proportion of images	$\frac{\# \text{ of images}}{\# \text{ of blocks}}$
Maximum header depth	the lowest relative depth of the headers. E.g. if a resource has <code>h1</code> and <code>h3</code> tags, then the maximum depth is two.
Average text size	$\frac{\text{total size of text in sentences}}{\# \text{ of text blocks}}$
Average code size	$\frac{\text{total size of code in lines}}{\# \text{ of code blocks}}$
Average table size	$\frac{\# \text{ of cells in table}}{\# \text{ of table blocks}}$
Average number of topics	$\frac{\# \text{ of distinct wiki topics}}{\# \text{ of text blocks}}$
Average number of task phrases	$\frac{\# \text{ of task phrases}}{\# \text{ of text blocks}}$
Average number of links	$\frac{\# \text{ of links}}{\# \text{ of text blocks}}$
Proportion of internal links	$\frac{\# \text{ of links to within the tutorial}}{\text{total number of links}}$
Proportion of external links	$\frac{\# \text{ of links to outside the tutorial}}{\text{total number of links}}$

with three or fewer characters while computing the size of a table, to avoid double-digit indices followed by a delimiting character. However, this technique disregards, for example, cells with regular expression characters.⁶ An alternative would be to build heuristics to match indexes in the resource set. Similarly, we chose not to consider tables when extracting task phrases and wiki topics. This avoids identifying briefly mentioned topics such as `Path`^(j) in the table in the Java Official resource “Object Ordering”.⁷ Any *relevant* topics should also be presented in the main text, and thus captured in our analysis. However, there exist some resources which format all information into tables.⁸ Pursuing perfect property extraction is time-intensive. In line with our goal to determine how to characterize resources, we focus our research effort towards investigating how the properties co-occur. Thus, although the descriptive statistics of the data set may be prone to noise, the property distributions show that there *are* variations in resource properties, suggesting the need for a framework based on properties to identify the design of a resource. We leave the improved and optimized extraction of properties to future work.

We applied Bonferroni correction when verifying the correlations between resource properties. However, this technique is conservative and may have resulted in missed significant correlations [49]. We supplement the analysis with the characterization of resources based on property co-occurrences. We retrieved traffic-related metrics (see Table 4.1) per website. Our preliminary investigation to retrieve and use resource mentions on Stack Overflow as a proxy for popularity of each resource revealed that only 981 resources (out of 2551) were referenced.^(k) Instead, a true representation of a resource’s popularity would involve accounting for proxy URLs, link redirections, parameters, and fragments of the resource’s URL. Due to practical limitations of computing this traffic accurately, we leave further investigation of the relation between design properties and resource popularity to future work.

Our framework for identifying resource styles relies on the deviations of properties. Thus, although our framework can be applied to any resource data set, it is better suited to ones that having large differences in property variations. This is because the differences between the resources will then be meaningful, and the styles can be perceived and verified within the data set. We leave the investigation of the success of applying our resource style framework on data sets with low variance to future work.

4.2 Resource Properties

To answer our first research question *to what extent do software technology tutorials vary in their structure and content-related properties?*, we investigated the variations in resource properties and the associations between them, for each programming language. We contribute a detailed view of properties of software documentation resources for five popular development technologies.

^(j)[http://en.wikipedia.org/wiki/Path_\(computing\)](http://en.wikipedia.org/wiki/Path_(computing))

^(k)We identified the number of mentions of the resource URLs in Stack Overflow answers present in the Stack Exchange Data Dump of June 2023.

4.2.1 Variations in Property Values

Figure 4.2 shows the distribution of resource property values by programming language. We use violin plots, which represent the frequency of resources (width) at different values (y-axis), with the median of the distribution indicated by a red line. For a given language, the leftmost violin plot shows the distribution of the number of blocks per resource. We see that for all resources except Java, the number of blocks in a resource varies to an upper limit of 500 blocks. For Java, the majority of resources also exist in this range, however one abnormally long resource exists with over 1500 blocks.

The next group of five plots represents properties that are proportions of different types of content per resource. Although programmers may seek code examples in software documentation, skipping corresponding explanatory text [210], only 7% (172) of the resources focus on code, i.e., contain more code than text blocks. Such resources demonstrate the usage of a particular component⁹ or describe *how* to write code to achieve a task.¹⁰ A total of 44% (1120) of the resources contain images, the type of element preferred by computer science students and professionals in information resources [69]. Resources use images to describe installation information,¹¹ architecture and modelling,¹² or to annotate the code with descriptions.¹³

The following marker plot in the figure shows the number of resources with a given maximum header depth. Each marker shows the proportion of resources with the corresponding maximum header depth value. We observe, for example, that C# and Python resources provide an overall more fine-grained organization than Java resources, most of which have at most two levels of headers.

The next six plots show the distribution of the corresponding properties from Table 4.3, providing an overview of the resource landscape. The median average code size (3-15 lines) is higher than the median average text size (1-2 sentences) for all languages. Despite the risk of large code snippets being difficult to understand [274], the resources have longer, but fewer code snippets compared to text blocks. Alternatively, although tables provide a large amount of information in a concise format, only 17% (421) of resources contain tables, with the largest table having a size of 213 cells.¹⁴

For all languages except C#, the distribution of the average number of task phrases is just slightly above zero. C# has a larger distribution (53 resources) with no task phrases. Considering task phrases as a proxy of procedural information [119], C# resources might not have instruction-like content.¹⁵

The last group of two violin plots in Figure 4.2 shows the proportion of the internal and external links in the resources. The distribution of the proportion of links shows greater density towards the top and/or bottom in the plots for all the languages. This indicates that, overall for a language, resources tend to contain either internal links to other tutorial resources or links to external resources, but rarely both.

To investigate whether there is a statistical difference between properties across programming languages, we performed Bonferroni-corrected [3] one-way ANOVA tests [82] ($\alpha = 0.05/15 = 0.003$ for each test). For the thirteen non-table related properties, we reject the null hypothesis, indicating that the grouping of resources by programming lan-

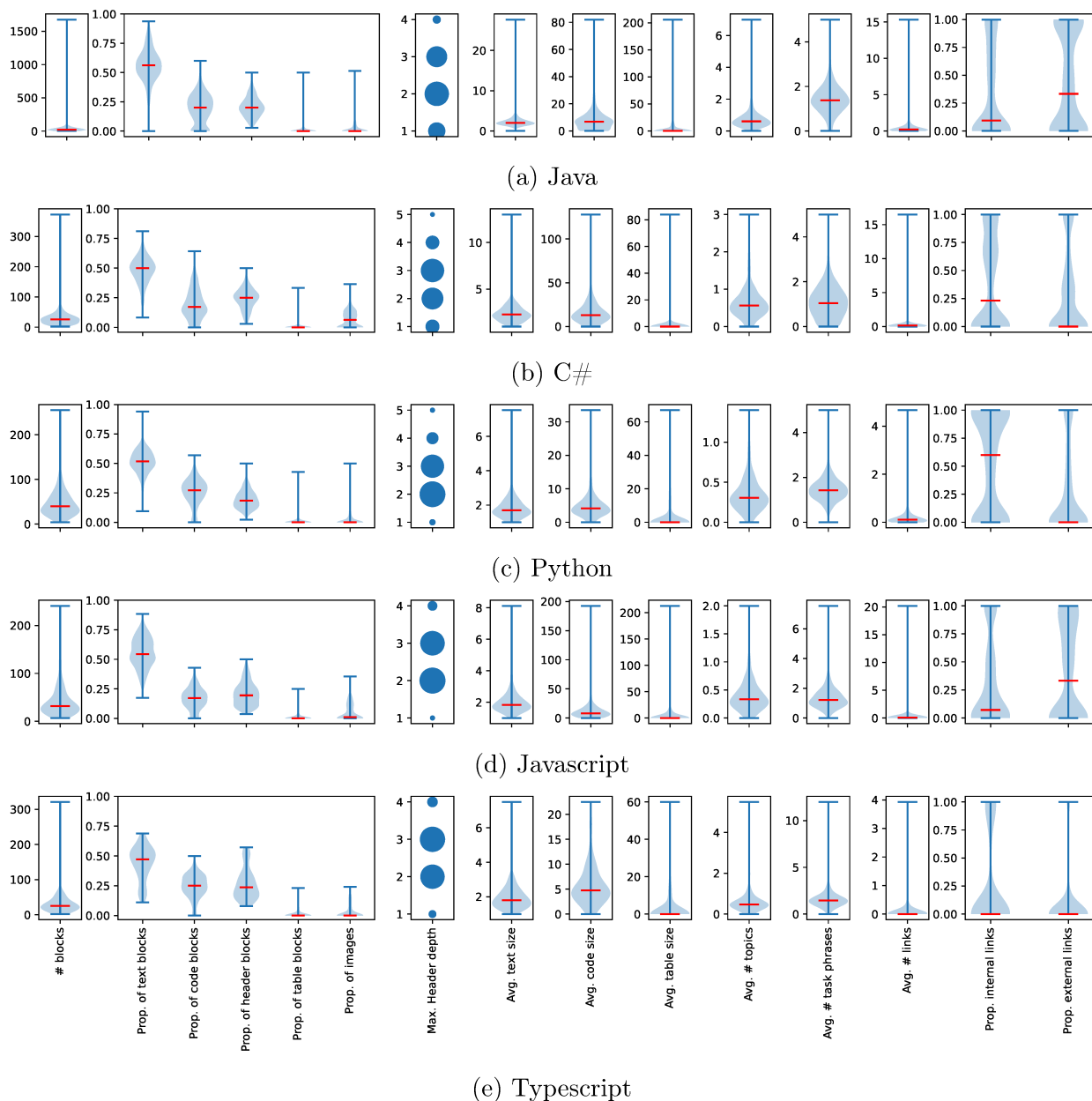


Figure 4.2: Variation of resource properties by programming language. The red line indicates the median of the distribution.

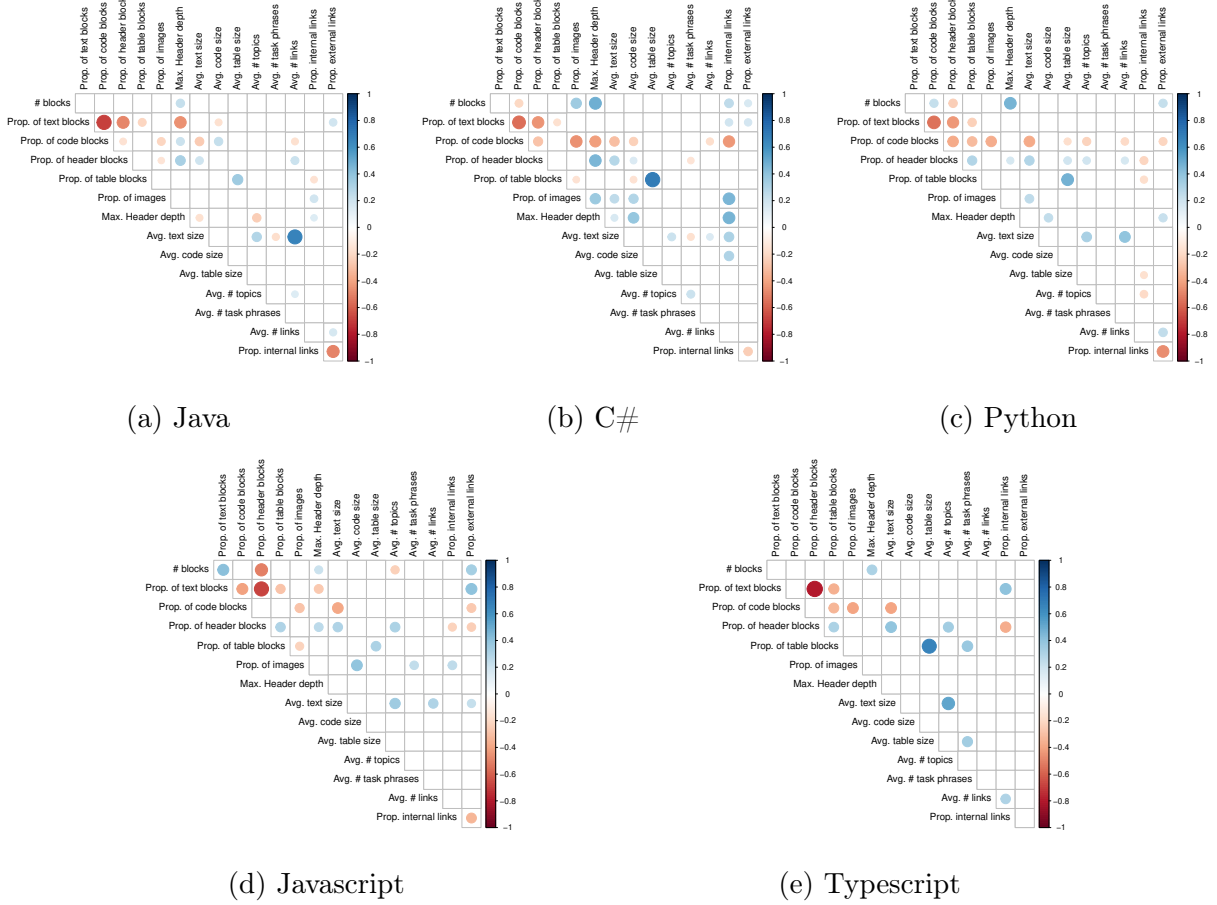


Figure 4.3: Correlation between properties for significant relations in each programming language. Only the significant results ($\alpha = 9.5 \times 10^{-5}$; $p < \alpha$) are shown. The colors correspond to Pearson’s correlation coefficient values.

guage accounts for a significant amount of the variation. However, we failed to reject the null hypothesis for the two table-related properties ($p = 0.149$ for the proportion of table blocks, and $p = 0.015$ for the average table size). The low variation of these properties may be attributed to the rare use of tables in resources (see Figure 4.2).

We also ran ANOVA tests to understand whether there is a statistically significant difference between properties across websites. We reject the null hypothesis for all properties ($p < (\alpha = 0.003)$), an indication that grouping resources by their host website can explain a significant amount of the variation observed. Prior work has studied the styling of code comments in Java and Python [203]; our findings suggest that further investigation into website-specific and language-specific tutorial design is worthwhile.

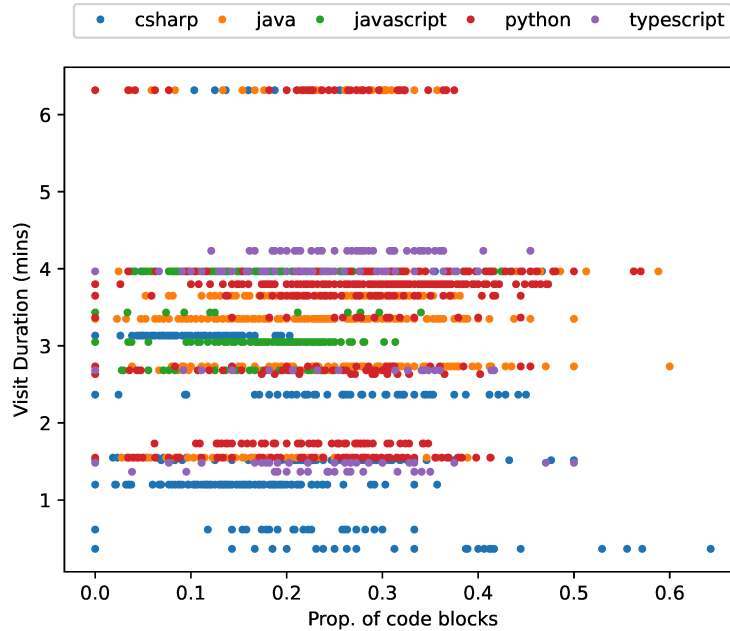


Figure 4.4: Distribution of the proportion of code blocks (from Table 4.3), against the average number of minutes per visit for the corresponding website.

4.2.2 Correlations Between Properties

We performed Pearson’s correlation tests to understand to what extent the variations in different properties are related. We performed 525 tests (105 tests for each of the programming languages), which introduces Type-I errors. To mitigate the errors, we applied Bonferroni correction [3]: $\alpha = 0.05/525 = 9.5 \times 10^{-5}$. Figure 4.3 shows the significant correlations between properties using the Pearson’s R metric. Each cell in the figure corresponds to a test. Blue markers indicate a positive correlation, whereas a red marker indicates a negative correlation.

For all languages except Typescript, there is a negative correlation between the proportion of text blocks and the proportion of code blocks. This is because the properties are calculated as a proportion of the total number of blocks, thus forming part of a complementary set. In C# and Typescript, there is a strong positive correlation between average table size and the proportion of tables in the resource. This may be because resources that contain more than one table typically summarize language keywords and their functionalities.¹⁶

There is a positive correlation between the average size of text and the average number of links in all languages except Typescript. For example, when introducing terms related to the main topic, Java Oracle documentation delegates the explanations to other resources.¹⁷ There is also a positive correlation between the average size of text and the average number of topics for all languages. Resources that introduce broad computing concepts such as data structures, link to other resources for detailed discussions of subtopics.¹⁸

4.2.3 Correspondence of Properties to Website Traffic

Software developers rate the inclusion of code examples and explanations as very important in API documentation [106, 165]. To determine whether the properties of a resource relate to user traffic, we analyzed the correspondence between each resource property we extracted and each website traffic metric (last three columns in Table 4.1). Figure 4.4 shows a scatter plot: each resource (a point on the plot) is mapped to the proportion of code in the resource (x-axis) and the average visit duration for the corresponding website (y-axis). The remaining pair-wise plots are available in our replication package.

Analysing the density and range of the distribution of points across the x-axis provides insight into whether certain property values may correspond to a particular level of traffic. We observe no clear trend that a certain proportion of code (or any other property) corresponds to a particular amount of time that a user spends on the website’s page (or any other traffic metric). This may be attributed to the fact that users have different, even contrasting, preferences about the design of documentation, which may be backed by some beliefs based on their prior resource seeking experience [16].

4.2.4 From Properties to Styles

We observe the open ended-nature and flexibility of tutorial content and organization. For example, resources have, on average, more text paragraphs than code fragments, and fewer tables and images. Furthermore, for the majority of property distributions, the density of resources is larger around the median. This indicates the existence of implicit normative values for resources for a particular programming language. Still, no property emerges as uniform across resources.

Resource creators can use the property distributions and correlations to identify gaps in existing tutorial offerings, e.g. visual elements that programmers prefer [69]. Resource seekers can leverage the properties to target resources whose content and layout is familiar or convenient to them. However, to do so, there is a need to characterize a particular resource *relative* to other resources, thus identifying it among the pool of varied designs. Furthermore, our observations of correlated properties suggest that dimensions for organizing a software tutorial [24] should not be considered independently. Rather, designing a resource requires deliberation about how different aspects of the resource complement one another.

4.3 Characterizing Resources

We answered our second research question *how can we systematically reason about the design of software technology tutorials?* by developing a framework for characterizing resources based on how they deviate from the norm for each of their properties.⁽¹⁾ For example, TutorialKart’s “TypeScript switch - Examples” resource¹⁹ is comparatively short, has longer

⁽¹⁾We explored the use of clustering techniques such as k-means to identify similarly designed resources. However, these techniques involved knowing, in advance, the number of clusters and were not interpretable for larger data sets. Instead, we focused on characterizing resources based on their co-occurring properties.

Table 4.4: Mapping of *Less* or *More* of a property (from Table 4.3) to an attribute of a resource.

Property	Inference of <i>Less</i>	Inference of <i>More</i>
# blocks	Short	Long
Prop. of text blocks	Text-light	Text-heavy
Prop. of code blocks	Code-light	Code-heavy
Prop. of header blocks	Contiguous	Fragmented
Prop. of table blocks	Table-light	Table-heavy
Prop. of images	Image-light	Image-heavy
Max. header depth	Flat	Hierarchical
Avg. text size	With-short-paragraphs	With-long-paragraphs
Avg. code size	With-short-snippets	With-long-snippets
Avg. table size	With-short-tables	With-long-tables
Avg. # topics	Topic-light	Topic-heavy
Avg. # task phrases	Non-task-oriented	Task-oriented
Avg. # of links	Link-light	Link-heavy
Prop. internal links	Not-cross-linked	Cross-linked
Prop. external links	Without-external-links	With-external-links

code snippets, and covers fewer topics than other Typescript tutorials in our data set.

To identify these deviations, we used quartiles of the property distribution across resources for the same programming language. We normalized each property against the maximum value of that property across resources such that all property values lay in the unit interval. The normalization allowed us to compare property distributions and better interpret the variations. We used the middle two quartiles of the property distributions across resources for the same programming language to define the norm of a property for a given programming language. For each property of each resource, P_r , we created two binary attributes: $less_P_r$ and $more_P_r$. We assigned $less_P_r$ as True if its value lies in the first quartile of the distribution and $more_P_r$ as True if its value lies in the fourth quartile. We referred to the properties along with their polarity as *attributes* and provide a mapping between the 15 properties and the corresponding 30 possible attributes in Table 4.4.

For every attribute, we determined its *deviation*, i.e. the absolute distance of the property value for the resource from the closest quartile. The deviation is given by $d(P_r) = |x(P_r) - q_i(P)|$ where x is the value of the property P_r for the target r^{th} resource, and q_i is the value of either the second or fourth quartile, as applicable, of property P . For each resource, we define the attributes that have any deviation for that resource, as *distinguishing attributes*.

Our observations from Section 4.2 suggest that treating each property of a resource in an isolated manner will result in neglecting the significant associations between them. Instead, understanding the design of resources requires investigation into how different properties co-occur in resources. We extend this observation to attributes. We propose the formalization of a *resource style* as a combination of distinguishing attributes of a resource. For

Table 4.5: Distinguishing attributes (d.a.) in the *prominent styles* ($n=3$) for all resources.

Nth d.a	1	2	3	4	Most freq. prominent style
Java	Cross-linked (0.22)	Flat (0.21)	Code-light (0.18)	Code-heavy (0.18)	Flat Text-heavy Code-light (0.036)
C#	Cross-linked (0.21)	With-external-links (0.20)	Code-heavy (0.20)	Contiguous (0.18)	Contiguous Flat Code-heavy (0.020)
Python	With-external-links (0.20)	Code-heavy (0.20)	Topic-heavy (0.17)	Code-light (0.17)	Long Hierarchical With-external-links (0.024)
Javascript	Cross-linked (0.23)	Image-heavy (0.21)	With-external-links (0.20)	Code-heavy (0.19)	Long Contiguous With-external-links (0.029)
Typescript	Cross-linked (0.24)	Image-heavy (0.20)	Table-heavy (0.17)	Fragmented (0.17)	Contiguous Text-heavy Cross-linked (0.034)

The number in parentheses refers to the proportion of resources in that language that contains the corresponding attribute(s) in its prominent style.

example, TutorialKart’s “TypeScript switch - Examples” resource mentioned earlier has three distinguishing attributes, which together form the characterizing resource style: *Short With-long-snippets Topic-light*. Since distinguishing attributes represent how a resource is different from others in its presentation, the style characterizes a resource’s design.

A resource style may be a combination of up to 15 attributes due to the mutual exclusiveness of attributes of opposed polarity (e.g., *Short* vs. *Long*). However, as the number of distinguishing attributes increases, the interpretability and practical significance of the style can decrease. We introduce three techniques to identify context-relevant resource styles. The *prominent style* acts as a resource’s identifier, providing resource seekers with a simple summarized interpretation of the design of the resource. *Recurring styles* provide insights on the current landscape of tutorials that can inform resource creators’ design process. *User-defined styles* allow both resource creators and seekers to systematically reason about the design of resources related to their own preferences. We motivate each with their practical use and discuss observations from applying each technique.

4.3.1 Prominent Style

The *prominent style* is the set of distinguishing attributes of a resource which *most* differentiate it from other resources.

Motivation: Although search engines reduce the search space, resource seekers are still required to make decisions between resources to determine which are more pertinent to their needs. Searchers use scents [194, 195] or cues [16] from meta information to find the information they need, however this may be implicit. Instead, a concise and explicit elicitation of the unique aspects of a resource can support the cue-following process, and comparison of resources.

Identification: The prominent style is the set of n distinguishing attributes with the largest deviation values. For example, the Python TutorialsPoint resource on sending emails²⁰ has five distinguishing attributes. However, with $n=3$, its prominent style is identified as

^(m)Based on the scaling of resource properties to binary attributes using quartiles, an attribute can occur in a maximum of 25% of resources. Thus, attributes can co-occur in a maximum of 25% of resources, since these resources are the set intersection of resources that contain each of the co-occurring attributes. In Table 4.5, the frequencies of the attributes are less than 0.25 because the prominent style for each resource is a subset of its distinguishing attributes.

Code-light, *With-long-snippets*, *Task-oriented* because these attributes have the largest deviation for the resource.

Observations: Table 4.5 shows the most frequent distinguishing attributes in the prominent styles for each programming language.^(m) For all the languages, the most frequent attribute is related to the links present in the resource. The deviations in the proportion of internal and external links is evident in the bulges shown in Figure 4.2 that lie on the end of the range, further away from the median. As another example, *Code-heavy* appears as a frequent distinguishing attribute in all languages except Typescript, because the distribution of the proportion of code blocks in Typescript is symmetric as opposed to the other languages (see Figure 4.2e).

4.3.2 Recurring Style

A *recurring style* is the set of distinguishing attributes that occurs among multiple resources for a programming language.

Motivation: Resource creators are faced with a number of design choices when creating resources [24]. To make informed choices about the content and organization of a resource, it is useful to assess existing resources [163]. The recurring styles provide an opportunity for resource creators to formally evaluate whether existing designs overcome organization-related issues that users have expressed regarding documentation [7, 207]. Subsequently, creators can choose to follow existing recurring styles or identify and fill relevant design gaps.

Identification: We use Formal Concept Analysis (FCA) [197] to uncover recurring styles across resources for a programming language. FCA is a framework for data analysis and knowledge discovery that is directly interpretable, and thus transparent and reproducible for a given data set.

FCA leverages an incidence matrix between *objects* (in our case, resources) and their *attributes* (from Table 4.4)⁽ⁿ⁾ known as the *formal context*,^(o) to explore latent relations. This exploration is supported by a hierarchy of *concepts* which are groups of objects, each called the *extent*, that share some attributes, i.e. the *intent*. The hierarchy allows for the subconcept-superconcept relation wherein the subconcept’s intent and extent are the superset and subset, respectively, of the superconcept’s intent and extent. The concepts are complete, i.e. all possible combinations of attributes occurring in the objects are identified. Our technique to identify recurring styles involves *attribute selection*, *formal concept selection*, and *identifying relevant concepts*.

a) *Attribute selection:* To mitigate the loss of interpretability as the number of attributes increases, we use variance thresholding [142] to select attributes^(p) with a non-zero variance. This method is based on the notion that features with the same value for all data provide

⁽ⁿ⁾The procedure we used to convert the resource properties to binary polarized attributes is an FCA technique known as conceptual scaling [105].

^(o)We used Python’s **concepts** API to build the formal context.

^(p)Prior attribute selection techniques are text-based [50, 51] and thus not applicable to our data. Others leveraging the distribution of objects [90] or distribution of attributes [44], rely on the size of the data set which would be biased by our attribute scaling technique. We also explored Correspondence Analysis [178], but found no clear representative dimensions.

no additional information to a data modeling algorithm.

b) Formal concept selection: The total number of formal concepts is a function of the number of objects and the number of attributes in the formal context. We perform context-specific^(q) concept selection [58] to identify important co-occurring attributes for resources. Our results from the correlation analysis (see Section 4.2.2) indicate that there exist significant correlations between properties, which can cause the constant co-occurrence of two attributes. For example, we observed that *Text-heavy* frequently co-occurs with *Code-light*. The formal context provides an opportunity to investigate more complex co-occurrences of multiple attributes, and thus we retain concepts with at least four attributes. Additionally, we retain concepts with at least five resources in the extent because the practical significance of a concept in our study decreases with fewer resources.

c) Identifying relevant concepts: We use *support* and *stability* metrics to investigate relevant concepts for each language. The *support*, given by the proportion of all objects that are in the concept’s extent, is a measure of frequency [110]. *Stability* is a measure of cohesion [129]: “A concept is stable if its intent does not depend much on each particular object of the extent.” [130] To calculate stability of each concept, we use the algorithm presented by Roth et al. [213] such that a value close to one indicates high stability. We use a threshold of 20 to identify concepts with the highest support and stability values,^(r) as *frequent, stable concepts* [110]. Because formal concepts are hierarchical in nature, the set of concepts obtained may contain subconcepts. Since our focus is on specific design differences between resources, we retrieve only the *maximal* concepts, i.e. those concepts which have no subconcepts within the selected set of concepts.

The *intent* of a maximal frequent, stable concept is the set of the co-occurring attributes that cohesively occur for multiple resources. We refer to the intents of the maximal frequent, stable concepts as *recurring resource styles*.

Observations: We identified between six and 14 recurring styles for each programming language. Table 4.6 shows the styles identified for Java and Python.^(s) Every row provides information about one recurring style. The first column of the table indicates the combinations of attributes (from Table 4.4) that form the style, e.g. *Short Contiguous Flat Text-heavy* is a recurring style for Java.

The identification of recurring styles provides insight into how the set of available resources are designed in the documentation landscape. For example, of the total 53 recurring styles, only three occur for more than one language:




















- *Text-heavy With-long-paragraphs Code-light With-short-snippets* (Java and Python) focuses on textual explanations, as opposed to code snippets, e.g. where code snippets only serve to demonstrate a topic.²¹
- *Short Fragmented Code-light With-short-snippets* (Python and Javascript) creates small

^(q)Note: context of our research, not formal context.

^(r)Jay et al. [110] used percentage thresholds to filter relevant concepts. We use an absolute threshold value of 20 concepts to avoid the dependency on the total number of concepts.

^(s)Our results of the intermediary steps and the recurring styles for other programming languages are available in our replication package and in Appendix B.2.

Table 4.6: Recurring Resource Styles in our data set for Java and Python resources.

Recurring Style	Stability	Support	NR	Websites	LNRW	NRWM
Java						Oracle
Short Flat Text-heavy Code-light With-short-snippets	1.0	0.075	59		57 (Oracle)	57
Short Contiguous Flat Text-heavy	1.0	0.057	45		44 (Oracle)	44
Text-heavy With-long-paragraphs Code-light With-short-snippets	1.0	0.071	56		46 (Oracle)	46
With-long-paragraphs Code-light With-short-snippets Topic-heavy	1.0	0.06	47		35 (Oracle)	35
Text-heavy Code-light With-short-snippets Topic-heavy	1.0	0.053	42		35 (Oracle)	35
Short Flat With-long-paragraphs Code-light With-short-snippets	0.99	0.06	47		46 (Oracle)	46
Python						Python Tutorial
Text-light Table-heavy With-long-tables Link-light	1.0	0.039	22		13 (JavaT-Point)	0
Fragmented Code-light Table-heavy With-long-tables	0.999	0.046	26		6 (W3Schools)	2
Short Fragmented Text-light Non-task-oriented	0.997	0.036	20		8 (Beginners-Book)	1
Text-heavy With-long-paragraphs Code-light Image-heavy	0.997	0.041	23		13 (Guru99)	0
Text-heavy With-long-paragraphs Code-light Topic-heavy	0.994	0.039	22		7 (Guru99)	0
With-long-paragraphs Code-light Image-heavy Topic-heavy	0.994	0.039	22		11 (Guru99)	0
Short Fragmented Text-light Link-light	0.991	0.03	17		6 (W3Schools)	0
Fragmented With-long-paragraphs Code-light With-short-snippets	0.99	0.039	22		8 (Guru99)	0
Text-heavy With-long-paragraphs Code-light With-short-snippets	0.989	0.037	21		8 (Guru99)	1
Fragmented With-long-paragraphs Code-light Topic-heavy	0.985	0.037	21		7 (Guru99)	0
With-long-paragraphs Code-light Image-heavy Link-heavy	0.985	0.036	20		17 (Guru99)	0
Short Fragmented Code-light With-short-snippets	0.983	0.03	17		6 (W3Schools)	1
Short Text-heavy Code-light With-short-snippets	0.981	0.03	17		4 (Beginners-Book)	2

NR — Number of Resources of the style,

Websites — The distribution of resources from different websites that correspond to the style,

LNRW — Largest Number of Resources of the style from a single Website,

NRWM — Name and Number of Resources of the style from the Website containing the Maximum resources for the programming language (see Table 4.1)

sections to address focused information, e.g. installation related pages.²²

- *Fragmented Text-light Table-heavy With-long-tables* (C# and Javascript) delegates some information to a table.²³

Although all the styles have a stability between 0.69 and 1, they occur in at maximum 7.5% of the resources for each language. This uniqueness among styles indicates that there is no natural progression towards a small set of common and distinct recurring styles, a possible consequence of the many design decisions taken during resource creation [24].

Whereas Python recurring styles are distributed across websites (column labelled ‘Websites’ in Table 4.6), we observe that many styles for the other languages are dominated by a particular website. This is the result of two compounding reasons, which we elaborate using the dominance of Oracle tutorials for Java recurring styles as an example. First, resources from the same website, in comparison to other resources, have similar distinguishing attributes. Figures 4.5a and 4.5b show the distributions of properties for Java Oracle and non-Oracle resources.^(t) Although the density of Java resources varies over a larger range (e.g. *proportion of text blocks, average code size*), the bulges tend towards one end of the range,

^(t)We removed the abnormally long resource TutorialsPoint Java Quick Guide²⁴ to make a comparison.

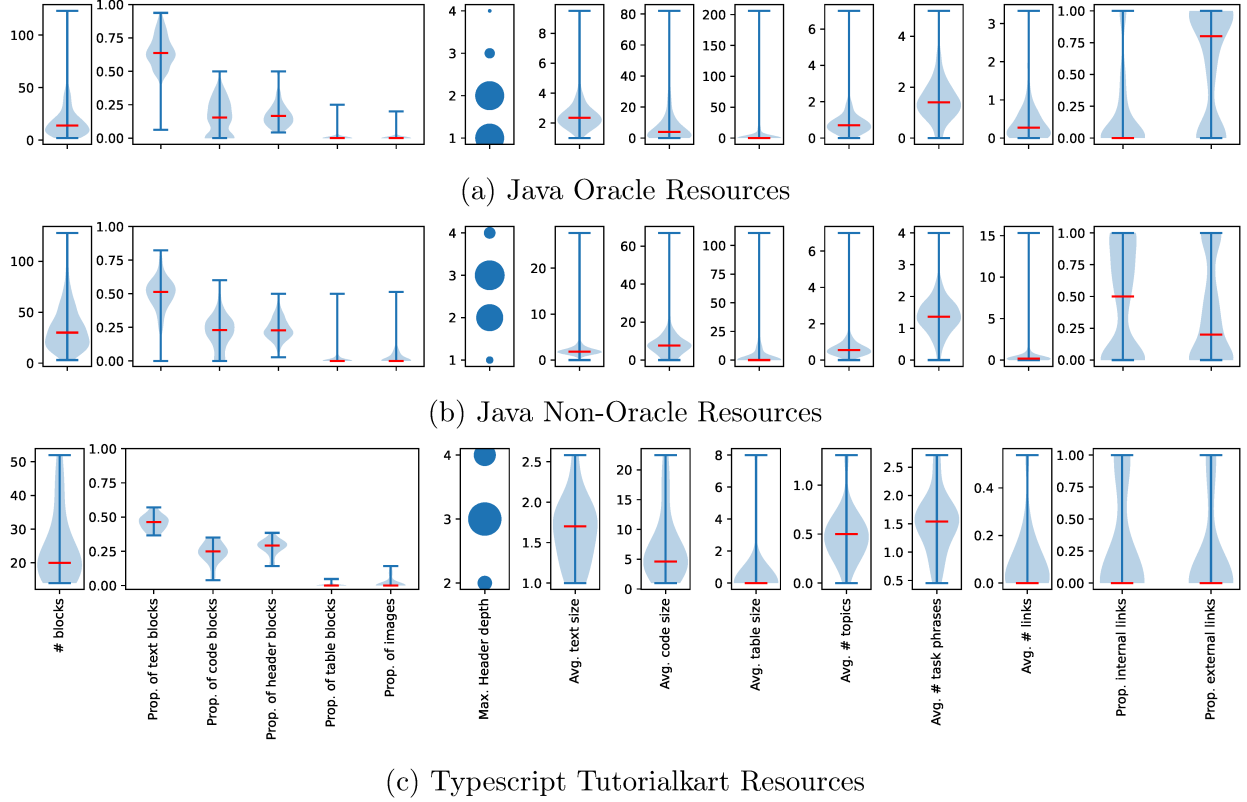


Figure 4.5: Variations of extracted properties in Java Oracle, Java not-Oracle, and Typescript TutorialKart resources. The red line indicates the median of the distribution.

corresponding to the first and fourth quartiles of the distribution. This abnormal behaviour is captured during concept scaling. Second, resources from the same website have frequently co-occurring distinguishing attributes. The most frequent recurring style, when identified for only Java Oracle resources, occurs in 57 resources. For non-Oracle resources, the most frequent style occurs in only 21 resources, indicating that more Java Oracle resources have co-occurring distinguishing attributes than non-Oracle resources do.

Recurring styles can also provide insight into website-specific design. We observe that TutorialKart is the only website for which no resources correspond to a resource style. This is because most properties of TutorialKart resources lie in the middle two quartiles of the distributions of all Typescript properties (compare Figure 4.5c and Figure 4.2e), and thus do not have co-occurring distinguishing attributes that are captured by the resource styles.

4.3.3 User-defined Style

A *user-defined style* is a combination of attributes explicitly selected by a user, e.g. a resource creator or seeker.

Motivation: The identification of prominent and recurring styles are helpful for reasoning about the design of resources, individually and in the context of other resources within the

software documentation landscape. However, creators may want to refer to existing resources that are designed similarly for inspiration. Resource seekers may already be aware about their resource design preferences [16, 62] during search. For both cases, the user-defined style enables the elicitation of attributes that are important to the user, to subsequently find corresponding, pertinent resources to the user’s needs.

Identification: A user states m attributes that are pertinent to their needs, e.g. a_1, a_2, a_3 . Then, we retrieve all resources for which these m attributes are distinguishing attributes.

Observations: The user-defined style allows users to retrieve resources that are most pertinent to their needs. For example, a resource seeker looking to solve specific problems with TypeScript can specify the style: *Short Text-light Task-oriented*. The resource seeker is presented with twelve appropriately designed resources, e.g. how to implement classes.²⁵ To explore further, they indicate *Code-heavy Topic-heavy*. Our technique retrieves eight resources that correspond to this style, and the resource seeker easily identifies one that provides more code snippets and covers a wider range of topics related to creating classes.²⁶ Thus, the user-defined style reduces the load on the user to manually identify which of the 172 Typescript resources correspond to their design preferences.

Similarly, a resource creator could intend to create a resource with a hands-on approach by providing more code snippets and visual elements like images. They specify *Code-heavy Image-heavy* to get an idea about existing resources. They find only four resources designed in this manner; this motivates the creator that their new resource can help fill the documentation gap for visual learning.

4.3.4 Discussion

The application of the proposed framework for characterizing resources on our data set demonstrates how the framework can be used to reason about resource design in a systematic manner. As a result, the framework supports the comparison of multiple resources based on their distinguishing design-related attributes. For example, both the C# GeeksForGeeks resource on the `switch` statement²⁷ and the C# TutorialsTeacher resource on the same topic²⁸ are *Contiguous*. However, whereas the latter is characteristically *Code-heavy*, the former is *Text-heavy*, *With long snippets* and *Cross-linked*.

The low frequency of prominent styles and lack of notable recurring styles indicates that there is no universal resource style: a challenge for automating tutorial creation. Instead, selecting what and how to present relevant information in automatically generated documentation depends on the information needs of a potential user given their task [211]. Our framework to characterize resources by their context-specific styles can be leveraged to inform the creation of flexible tutorials.

4.3.5 Limitations

The resource set in this study is a convenience sample based on popularity of technologies, and ranking by the search engine DuckDuckGo. With the variation of properties for a website, we concluded that using stratification techniques to balance the data set with a

proportional number for resources from different websites would result in a misrepresentation of resources available online. We observe that styles are influenced by similar co-occurring attributes, irrespective of the website. For example, despite JavaTPoint resources making up 57% of Javascript resources, not all recurring styles occur in JavaTPoint resources. As a result, we deliberately disregarded the website during data analysis, and report our observations treating each resource equally as an independent web page.

Our observations of resource styles in 2551 resources demonstrate that our framework can provide interpretable and useful insights about resources. We focus on the *design* of resources, and thus disregard the technical topics the resources cover when applying our framework. We propose the framework as a way to characterize resources about similar technology topics, and assume that topic pertinence is handled separately and parallelly to the identification of the resource's design. Our design analysis framework may be expanded to include properties identified in other tutorials, or even other documentation types. Furthermore, the framework may also be applied in other use-cases with new, more appropriate techniques to identify other combinations of co-occurring attributes as resource styles. We also provide the necessary data to investigate variations of properties and styles within websites and across programming languages.

Chapter 5

Considerations of Documentation Creators

Documentation creators have a variety of decisions to make about content and presentation [24]. These different decisions can result in variations in the style of documentation (see Chapter 4) [19]. To support information seekers in navigating among varying resources, prior work has focused on improving the efficiency of search [8, 111]. Higgins and Scholer explained that to understand the value of any outcome, such as the achievement of a task, it is also important to understand the *process* of generating that outcome [93]. Thus, understanding the *context* in which the documentation was *created* can provide insight on the design of documentation. The design context can then guide information seekers towards resources pertinent to their needs.

We interviewed 26 documentors to understand their motivations and techniques to create and contribute documentation online. We use the term *documentor* to refer to someone who voluntarily creates and contributes online documentation.

Goal

The goal of this phase of the research was to understand documentation creators' considerations when voluntarily creating and contributing software documentation online, despite a wide variety of existing resources.

Research Question

Why and how do people voluntarily contribute documentation online?

Publication

The study of why people voluntarily contribute software documentation online was published in the article *Why People Contribute Software Documentation* [22]. The extended work including the investigation on what the documentation process entails is available as an ArXiv preprint titled *The Software Documentor Mindset* [21].

Replication Package

The coded data set as well as the documents needed to replicate the study are available in our online replication package: <https://doi.org/10.5281/zenodo.14416777> [20].

5.1 Study Design

We conducted semi-structured interviews, which we subsequently analysed using card sorting [102] to gain a better understanding of the software documentation contribution process.

5.1.1 Informant Recruitment

We directly invited documentors, instead of having an open call for participation, to ensure that informants had regularly and recently contributed documentation. We focused on people who released blog articles or YouTube videos about some software technology. Although we began our search through documentation for Java and Python, we did not disregard documentors of other technologies. Popular blogging websites such as medium.com and hashnode.dev do not provide a standard method to contact bloggers, and contact information such as email was rarely provided. Instead, we recruited the first informant via personal contacts, and used different techniques to subsequently identify documentors:

Github: For each of Java and Python, we used the Github API to retrieve repositories that were in English, and contained both the name of the technology and the word ‘tutorial’ in either the name, description, or README of the repository. To recruit more informants, we further expanded our query to C++, Ruby, and SQL.

YouTube: For each of Java and Python, we manually searched for the following queries in the search engine DuckDuckGo, in the video tab, in incognito Chrome browser:

1. <technology> tutorial
2. <technology> programming tutorial
3. <technology> development tutorial

and retrieved each of the search results from the first three pages of the queries.^(u) In total, we obtained 409 unique video links for Java and 421 unique video links for Python.

For each of the Github and YouTube search results, the author of this thesis manually determined if the contributor was an individual, i.e. not a community of creators or a company. The author also confirmed that the contributor had contributed documentation related to the working and usage of a software technology, irrespective of the technology they were

^(u)We used the common term ‘tutorial’ to identify instructional resources, as opposed to other forms of documentation such as reference documentation.

documenting, within the past six months, and at least a total of three times.

WriteTheDocs: WriteTheDocs is “a global community of people who care about documentation” [266]. The community has a Slack workspace in which technical bloggers can introduce themselves in the Slack channel `intros`, and share their work in the channel `community-showcase`. Between January and April 2023, we monitored both of these channels and invited bloggers who had created a post in the past two months about the working and usage of software, and had created at least three blog posts so far.

We obtained the publicly available email IDs of the identified contributors and sent them an email inviting them for an interview. Table 5.1 shows the details of the 26 informants we recruited.^(v) Informants were not monetarily compensated for participating in the study. The study protocol was approved by the Ethics Review Board of McGill University.

5.1.2 Data Collection

Since our goal was to understand *why* and *how* people contribute software documentation, we performed semi-structured interviews [137], as opposed to structured interviews based on a pre-defined set of questions. This approach gave us the opportunity to gain context-specific insight based on the informant’s experiences. We asked informants about their journey into documentation and their process of creating documentation, including how they selected topics, and what their procedure for creation was. We asked informants to expand upon aspects that they would bring up, allowing them to steer the conversation according to their documentation experience. We incorporated any new aspects, such as whether and how they announced their newly released documentation, in interviews with later informants. Our interview guide is available in our replication package and in Appendix C.1.

5.1.3 Qualitative Analysis

We qualitatively analysed the interviews in a manual, iterative manner. After completing the interviews with the first five informants, the author of this thesis began open coding the transcripts. The open coding process resulted in identifying three major dimensions of the documentation process. *Motivations* describe the incentives for contributing documentation. *Topic selection techniques* capture how to decide on topics to create documentation about. *Styling objectives* describe the purpose of and rationale behind content and presentation design decisions.

The three dimensions occur differently during the documentation process. Motivations act upon the documentor and *encourage them to contribute documentation*. In contrast, *topic selection techniques* and *styling objectives* are dimensions that the documentor decides

^(v)We interviewed two additional documentation creators, recruited via references of two informants. However, one creator created documentation for their own software in the form of linux man pages, and the other created documentation for online courses that were not publicly available. We do not report on these two interviews, as they are out of scope of our focus of publicly-available online software documentation.




CHAPTER 5. CONSIDERATIONS OF DOCUMENTATION CREATORS

Table 5.1: Details of the informants of the interview study, all of whom are documentors.

	Recruited from	Type of content	Programming exp. (yrs)	Documentation exp (yrs)	Familiar technologies
P1	Reference	Text	25	13	Javascript, Python
P2	Github	Text	40	10	C++, Python
P3	Github	Text & Video	25	6	Java, Kubernetes
P4	Github	Text	33	9	Python
P5	WriteTheDocs	Text	5	1	Python
P6	Github	Text	24	1	Python
P7	Github	Text	1	0*	NodeJS, Cloud
P8	Github	Text & Video	23	3	Python, Rust
P9	WriteTheDocs	Text	4	2	Javascript
P10	YouTube	Text & Video	22	17	Java, Spring
P11	YouTube	Video	5	3	Python, Plotly, Dash
P12	Github	Text	21	13	Asp.net, C#, HTML
P13	WriteTheDocs	Text	3	2	NodeJS, ReactJS
P14	WriteTheDocs	Video	5	3	Git
P15	Github	Text	20	17	PHP, Javascript, Python, Go
P16	Github	Text	16	9	HTML, CSS, Javascript
P17	YouTube	Video	10	5	Python
P18	YouTube	Video	8	5	Java
P19	WriteTheDocs	Text	9	2	Python, Docker, Git
P20	Github	Text	9	4	SQL, C++, Python
P21	Github	Text	12	4	Python, GNU/Linux
P22	Github	Text	2	0*	Python, C++
P23	Github	Text	8	4	Javascript, Typescript, NodeJS
P24	Github	Text	25	10	Javascript, Typescript, web dev.
P25	YouTube	Video	12	3	C++, Java
P26	Github	Text	10	7	C++, Python

+ If the informant did not self-report the extent of their experience during the interview, we retrieved this information from their LinkedIn profile or public documentation. The reported experience is as of when we interviewed the informant, and is rounded to the nearest year.

* 0 indicates that the informant's experience was less than six months at the time of the interview.

upon during the documentation process. Thus, the documentor *designs their documentation through* these two dimensions. We use  for motivations,  for topic selection techniques and  for styling objectives, to indicate when and how each dimension manifests during documentation contribution.

For each dimension, the author of this thesis performed card sorting [102] of the open-coded responses from the informants, to identify documentors' *considerations* during the documentation process. Then, all the authors, i.e. the author of this thesis and her supervisors, together discussed the relevance, meaningfulness, and accuracy of the identified *considerations* to mitigate researcher bias. We identified five motivations, five topic selection techniques, and six styling objectives. We describe documentors' considerations across the three dimensions, in detail, in Section 5.2.

5.1.4 Mindset Elicitation

We noted that different considerations can be thematically related across dimensions. For example, P3 described their motivation to contribute documentation: "I wanted my examples to be available for everybody to be used and that's why I started putting content on GitHub." [P3] They also described that they selected topics based on popularity: "a lot of people should be interested in that topic." [P3] However, we could not reliably infer that P3 selected popular topics *because* they wanted their examples to be used by more people. Still, P3's motivation and topic selection technique are both related to reaching an audience. We grouped such related considerations

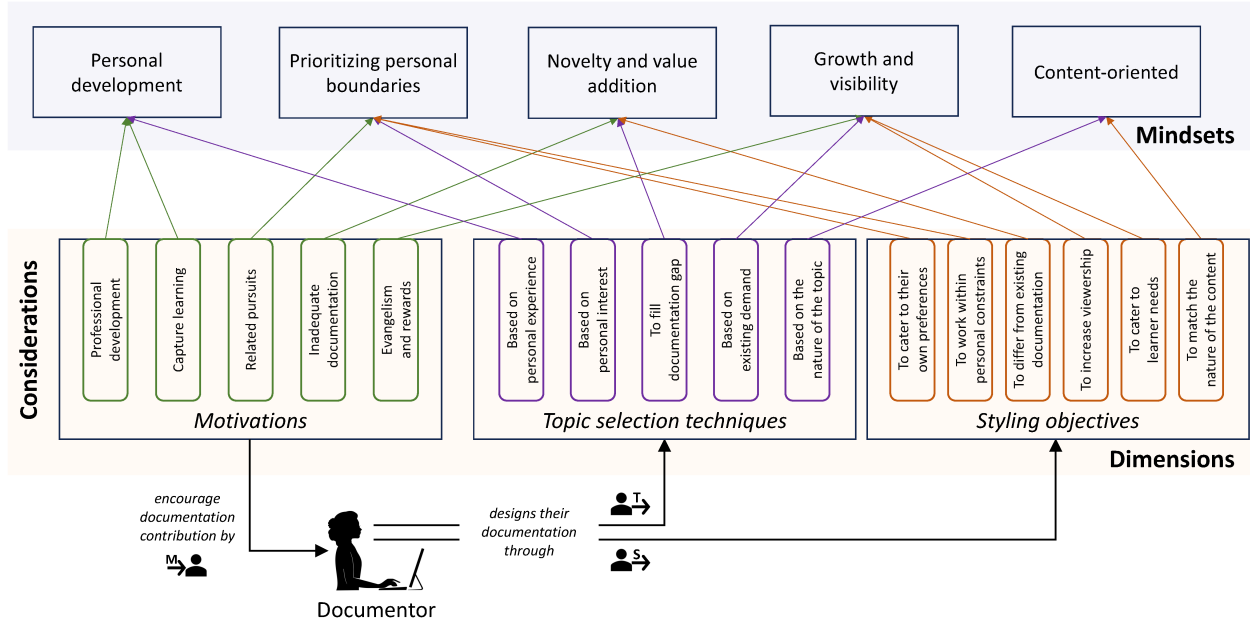


Figure 5.1: Framework of documentors' mindsets and their associated considerations across the three dimensions of the documentation contribution process, i.e. *motivations*, *topic selection techniques*, and *styling objectives*, based on interviews with 26 documentors.

and thereby elicited five *software documentor mindsets*. The mindsets provide an overview of what documentors think about when contributing documentation. We propose the mindsets and their associated considerations across three dimensions as a framework for understanding why and how people contribute documentation, as seen in Figure 5.1. We describe the mindsets, in detail, in Section 5.3.

5.1.5 Validation

To verify whether the mindsets credibly capture informants' thought processes during the documentation process, we performed member checking [53, 223]. We sent informants a questionnaire with a preliminary, one-line, description of the mindsets. We asked respondents to select to what extent they agreed to having experienced that mindset, on a four-point agreement scale with an additional "Unsure" option. If they agreed, we asked them to briefly describe how they experienced the mindset. At the end of the questionnaire, we asked respondents if they had experienced any mindsets that were not captured by the five we elicited, and whether they had any additional comments. Whereas one email invitation bounced, we received a total of 17 responses, which we discuss in Section 5.4. The validation questionnaire is available in our replication package and in Appendix C.2. The open-coded data is available in the accompanying online replication package.

5.2 Dimensions of the Software Documentation Contribution Process

Our analysis of the interviews revealed *considerations* informants had while contributing documentation, across three *dimensions* of the documentation process: *motivations*, *topic selection techniques*, and *styling objectives*.

5.2.1 🗣️ Motivations

We elicited five major motivations that capture why our informants contributed software documentation (see Table 5.2).

Professional development

Informants expressed the importance of their documentation as an online portfolio of their knowledge for potential recruiters (13 informants). For example, documentation provided an opportunity for informants to keep up with technologies of interest that they were not exposed to in their regular employment (P23). Informants also received external incentives to contribute documentation, such as encouragement from their employers (P1, P10, P15, P20). When the informant represented the company publicly, the created documentation provided evidence of their authority: “We’re also encouraged to have our own brand and my boss wants me to be out there promoting my brand so that when we’re at conferences, people look to me as an authority, like this person does know what they’re talking about.”[P10]

Capture learning

Informants described that their documentation was like a systematic set of notes (P8, P15, P25), which they could refer to in the future (P16, P21): “If you have to work hard to find something out by pulling lots of things together, then putting it in one place in your blog is cool. Then the next time I can use those instructions. So it’s just notes, but public.”[P15] This was prompted by frustrations of forgetting useful information (P16, P24): “I wanted to have an archive for myself that I could consult. I had the experience where I would run into some problem, spend three hours fixing it, and then I would move on to the next thing. A month later, I would have the same problem and I wouldn’t remember how I fixed it, so I’d have to spend another three hours rediscovering the solution.”[P16] Ultimately, creating the documentation was a way to help understand a topic better (eight participants): “When you have to explain something to someone, then whether you can *actually* explain shows that you understand something.”[P5]

Related pursuits

For some informants, creating documentation was the result of combining programming with another personal pursuit, e.g. programming and teaching (P17, P19), video creation (P18, P25), or writing (P2, P4). “I wanted to make a YouTube channel because me and my friends from high school would make YouTube videos for video games. [...] I thought, well I have this programming experience [...]

Table 5.2: Documentors' considerations along the dimension *motivation*.

Consideration	Description	Example open code
Professional development	Informants described that they were contributing documentation as a portfolio to demonstrate their knowledge to potential employers and customers.	Good marketing (of self) as a freelance consultant
Capture learning	Informants created documentation about what they were learning, to understand the technical aspects of software better. The documentation also acted as a repository of information that the informant could refer to in the future.	To understand the topics, by explaining to "someone else" via the documentation
Related pursuits	Informants created documentation because they were curious about what it involved, or because they had related interests such as teaching.	Connected passion for writing with technical knowledge as "own art"
Inadequate current documentation	Informants created documentation to overcome the issues they faced with documentation when learning or searching about technical topics.	Existing documentation has too many details [for a beginner]
Evangelism and rewards	Informants created documentation to help others (altruism) or to gain benefits (e.g. monetary compensation).	Put up [documentation] content to get minimum level of income
Other motivations	Informants described other motivations, such as being inspired by authoritative people to contribute documentation, or wanting to receive feedback.	Tutor at boot camp mentioned technical writing so decided to try it

and I ended up deciding to make programming tutorials.”[P18] Documentation also provided a way for informants who had struggled while learning programming to utilize their knowledge and stay connected to programming (P5, P9). P5 expressed that although they were interested in learning programming, they did not think they would ever get to the level of a full-time programmer. Instead, blogging would allow them to still get an idea of how things work. Similarly, P9 explained: “I just felt like: I’m really struggling with coding. I don’t feel fulfilled when I write code. What about just trying out technical writing?”[P9]

Inadequate current documentation

When the documentation was lacking (P12, P19), inaccessible to beginners (P4), overwhelming (P5), or scattered across multiple resources (P23), informants felt the need to fill the gap with relevant documentation. P12 explained: “That’s really how I ended up writing: I found that an awful lot of the documentation online was either nonexistent or quite obtuse. It was difficult to read from a beginner point of view. So I try things and then write an article about it and put it on my blog.”[P12] Informants described that existing documentation did not cater to their preferences [16],

prompting them to create pertinent documentation: “I figured that a lot of people would appreciate recorded video material because that’s what I appreciated. And I saw a gap there.”[P11] Informants could then tailor their documentation to their needs: “[I decided] I will start a blog where I will write content the way I wanted people to write them, when I was still learning.”[P23]

Evangelism and Rewards

Informants felt the need to “give back” [155]: “I had learned a ton of stuff on YouTube throughout my educational journey [...] So I was like - I’m going to teach people on YouTube.”[P17] They were motivated to help other people gain access to knowledge: “I wanted my [code] examples to be available for everybody and that’s why I started putting content on GitHub and wherever possible writing articles, so other people can access the content.”[P3] Although monetary compensation can drive documentation creation (nine informants), informants explained that it did not always reach their expectations (P8). Yet, other benefits were motivating, e.g. positive comments from users were encouraging (seven informants): “[When] I started the YouTube channel, the videos did receive some good feedback and I just decided to continue with a few more topics that I liked. And then this was all coming together and I started doing more and more videos on the channel.”[P25]

Other motivations

Informants described other factors, although less notable than the previous five categories, that motivated them to contribute documentation. For example, someone else advocated for creating documentation (P6, P7, P16, P21): “It was from an article [...] the author] mentioned technical writing. And also from my program in a bootcamp, [...] a tutor came and talked about it. [...] That’s why I just decided to try and write.”[P7] Informants (P10, P18) wanted “something tangible that I could build that was my own.”[P10] Informants also wanted to build a network (P26) to obtain feedback [189], and improve other skills like writing and English language skills (P21).

5.2.2 ➡ Topic Selection Techniques

We elicited five techniques informants used to decide what topics to create documentation about (see Table 5.3).

Based on personal experience

Informants described that they wrote about topics that they were familiar with, for example, technology that they had recently learned about in their daily work (21 informants): “My day job is in Python, so if I’m doing something useful and it’s interesting at work, I may spend an hour in the evening documenting.”[P8] This meant that the informants were already familiar with the topic: “I thought that it could be a nice idea to review all the basics and share the knowledge I already have.”[P22] An advantage of this technique is that the content comes from experience with a real application: “I basically baked down all of the things that I did [at work] and put it into that video. So it was very much real world experience baked into a video.”[P17]

Based on personal interest

The voluntary nature of contributing documentation allowed informants to explore topics that they were curious or passionate about (15 informants): “Most of the time, these were topics that I personally liked and wanted to investigate more. I already worked with micro-controllers, but I had never gone into every single detail of it.”[P25] Contributing documentation also provided the opportunity to go beyond their professional experience: “I purposefully picked technologies I would not have any exposure to at my job, but that I’m interested in and I don’t want to wait around to have to learn.”[P6]

To fill documentation gap

Informants selected topics for which they found the existing documentation either too complex for beginners, not extensive, not to their preferences, or completely lacking (seven informants): “People were explaining how [something] was working, but they weren’t showing how they did it.”[P22] They focused on difficult topics to document, as there would be fewer resources on such topics (P6, P17, P19). Traffic analytics, such as failed searches for what topics people wanted to learn about but did not find (P1), and volume of existing documentation on a particular topic (P2) were helpful to make informed decisions about what topics to cover.

Based on existing demand

Informants selected topics because there was a clear demand from the audience for particular topics. They monitored community channels and question forums (six informants): “I see what people ask because usually I can answer them. Or I can tell them to search my blog or my website and I know there’s a solution there. Sometimes the questions come up and I cannot answer them right away, or the answer would be something larger and I have to try it out myself. And that gives me a topic for a blog post.”[P24] Informants also directly asked their audiences what topics they wanted (P9, P10, P20): “In one of the newsletters, I asked them if they want me to cover some specific topic. They can just send me a message about it, and I will do it.”[P23] Additionally, informants also selected topics that, from their experience, they knew people would want (P1, P7, P12), and would get more views (P8, P10, P18, P25).

Based on the nature of the topic

Informants considered how suitable a topic is to documenting, e.g. by catering to particular audiences (P1, P15), such as beginners (P2, P4, P5), or selecting topics that are applied and hands-on (P2, P14, P17, P26). Informants developed topics towards an actionable deliverable (P17, P22): “Selecting the actual topics was like building off of what I had already [documented]. I built my first game tutorial after making five introductory Python lessons that cover the basics to build that game. So, I was building the blocks... Now I can make a game.”[P17] Whether the topic is “documentable” is important: “One step should be self-explanatory, so you go step by step. And so that’s why I’ve kind of moved away from those other [topics] that would be complicated.”[P5]

Table 5.3: Documentors’ considerations along the dimension *topic selection technique*.

Consideration	Description	Example open code
Based on personal experience	Informants decided to write about a topic that they had experience with, e.g. a topic they worked on at work, or while learning.	Solutions to issues faced at workplace
Based on personal interest	Informants selected topics that they were curious about and/or that interest them.	Learning [topics] to support kids’ interests
To fill documentation gap	Informants selected topics for which they did not find documentation or the existing documentation did not explain the topic well.	If a topic may benefit a lot of people and has not already been covered well on YouTube
Based on existing demand	Informants selected topics that many people would want to learn about.	Multiple people asked the same question online, so create blog post to refer them to
Based on the nature of the topic	Informants selected topics that have a particular “documentability”, e.g. they are appropriate for beginners, or are easy to create a tutorial about.	One [topic] that is slightly obscure that can be justified well by an article
Undefined	Informants described "naturally occurring" ideas or that they did not know exactly how the ideas for topics came to them.	May come from a variety of ideas - no specific pattern

Undefined

For some informants, ideas for topics manifested naturally (P1, P2, P5, P16). “There’s not a clear pattern [...] I’ll be walking down the street and I’ll have an idea for something that I think could make a good blog post. Or even sometimes it is really granular, like an interactive kind of visualization I have in mind for a particular way to describe some particular concept. And so I jot down whatever notes I have into Notion [a note-taking tool]. And after a while, if the same idea keeps coming back to me, I figure it’s probably a good idea to actually write about it.”[P16]

5.2.3 🧑🔧 Styling Objectives

We elicited six objectives that informants had for the organization and presentation of content (see Table 5.4).

To cater to their own preferences

Informants created documentation the way they like documentation to be (seven informants): “I actually don’t really enjoy reading that much, because there are a lot of words. I like reading simple things, so this is why I try to make [my documentation] simple.”[P7] Informants acknowledged that they focused

on what worked best for themselves: “[In my documentation,] I always like to follow [the structure:] the situation, the problem, and the solution. Because for me - and I’m definitely biased - this is the way I learn.”[P23]

To work within personal constraints

Informants described how they styled their content based on what was convenient and feasible to their time and effort (P2, P4, P11, P14, P18, P21). For example, to determine what code examples to add, P2 described: “It has to be code that’s simple enough that I can write about it in a relatively short time.”[P2] Informants modified their documentation accordingly: “If it’s taking me too long, I just try my best to wrap it up, don’t put anything else. Just put a link to the original articles or some Stack Overflow questions and answers and finish it.”[P21]

To differ from existing documentation

Informants strategically styled their documentation to stand out from existing documentation (P8, P16, P17): “There’s a thing called [technology name]. It’s basically a wrapper where you can paste in your code and then it creates permalink. I put the permalink in my video description. Not everybody does that in their video, so I’m hoping that makes it a nice feature of my channel.”[P8] Similarly, P16 added interactive components in their textual blog, to dynamically see the impact of code changes: “Most blogs have some sort of static format like Markdown. [...] So the goal with my blog was to be able to create these one off components [...] that can be then embedded in the blog post.”[P16]

To increase viewership

When designing their documentation, informants added features that would capture the attention of their audience (nine informants): “Your [article] title is really really important. Because at the end of the day, you want to have a title that when someone searches it on Google, it might pop up.”[P9] Additionally, they optimized for search engines to index the documentation on earlier pages (P1, P9, P18, P23, P25): “I’m also trying to add the YouTube chapter markers to videos. YouTube is promoting videos better if there are chapter markers in there.”[P25]

To cater to learner needs

Informants were cognizant of the information and styling needs of users when designing their content. For example, documentation needs to be very clear and concise for the audience (nine informants): “In the software world, developers follow some technical jargon. I avoid using them because I need to explain to an audience and it should be really clear.”[P19] Informants deliberately styled their content to help the audience understand better (ten informants): “If you are new to this area and some people are explaining things really, really quickly and you just have a big chunk of a code snippet at the end, [...] you are just like: OK, so what does this do, why this, why that. That’s why, [in my documentation,] every single step has a code snippet rather than just one at the end.”[P5] To assist the audience, informants provided useful indicators about the content (P13, P16, P20, P24, P26), such as a note about prerequisite information requirements: “[The readers] need some kind of knowledge before, to better understand the article. So I always make sure I let them know what they need.”[P13]

Table 5.4: Documentors' considerations along the dimension *styling objective*.

Consideration	Description	Example open code
To cater to their own preferences	Informants incorporated aspects that they would have liked to have as a learner.	Is participant's personal preference to have a structure with situation, problem, and solution
To work within personal constraints	Informants styled their content based on how much time they had, or to keep an achievable routine for posting documentation.	Code has to be short enough that it can be written about in a relatively short time
To differ from existing documentation	Informants took deliberate measures to ensure their documentation was different from existing documentation about the same topic.	Differentiate from existing static blogs by providing interactive real examples
To increase viewership	Informants used techniques to capture a learner's attention and optimize for search engines.	Keep reader engaged and incentivized to read until the end, by adding humor
To cater to learner needs	Informants styled content to what they thought a learner would want or need.	Have three examples so at least one may be relevant / valuable to the viewer
To match the nature of the content	Informants styled content based on what suited it best, e.g. examples are well suited to explaining hands-on topics.	GUI software lends itself better to screen-shots/images
Other objectives	Other styling objectives that are less notable.	Follow mentor's documentation as a guide

To match the nature of the content

In some cases, the content is naturally well suited to a particular type of presentation (P1, P3, P4, P15, P17), and the informants leveraged this characteristic when creating their documentation. “The type of software that I work on mostly tends to be data visualization or data analysis software, which lends itself very well to nice canned examples that go from top to bottom.”[P1] Informants styled their content based on whether they wanted to cover a breadth of topics or wanted to go in-depth: “Over an one-hour time period, I can cover a huge range of topics [by providing hands-on examples], which would not be possible if I [document] in a pictorial way.”[P3] Informants covered the most necessary details to make their documentation self-contained (P24), and provided an appendix (P21) or links to additional material (P15, P25), to avoid overwhelming readers.

Other objectives

Informants described other reasons and goals for styling. For example, informants followed best practices: “A basic concept in technical writing is [covering] a single topic. So you should stick to one thing [in a single document].”[P5] They were also inspired by other resources: P16 saw other documentation have a note with the intended audience, and incorporated the idea in their own documentation because they found it valuable.

5.3 Software Documentor Mindsets




We observed that different considerations can be related by common, implicit themes. For example, the  motivation *lack of inadequate documentation*, the  topic selection technique *to fill documentation gap*, and the  styling objective *to differ from existing documentation* are all related to contributing new and improved documentation. The author of this thesis identified and grouped considerations by the underlying themes. We obtained a total of five groups, which all the authors discussed based on evidence from the interviews. The groups capture what documentors think about during the documentation creation process, across the three dimensions. Thus, we identify them as *software documentor mindsets*.




Figure 5.1 shows the framework of the five mindsets observed among our informants, and the corresponding considerations across the three dimensions of the documentation process. These elements of the framework impact documentation creation differently.  *Motivations* encourage people to contribute documentation, whereas the documentor designs their documentation through  *topic selection techniques* and  *styling objectives*. We mapped the mindsets for each informant, by identifying whether the informant described the associated considerations. The mapping, a sample of which is shown in Table 5.5, documents how the mindsets manifest for each informant. For example, for P5 we identified multiple considerations associated with the *personal development* mindset. In contrast, P1 had this mindset when they were initially motivated to contribute documentation, but more prominently had the mindset of *growth and visibility* when selecting topics and styling their documentation. We describe the five mindsets with examples from our interviews.

Table 5.5: Documentors’ mindsets and the corresponding considerations across the three dimensions of the documentation contribution process, for each informant.

Mindset	Dimension	Consideration	P1	P2	P3	P4	P5 ...
Personal development	👤 Motivation	Professional development	X	X		X	X
	👤 Motivation	Capture learning					X
	👤 Topic selection techniques	Based on personal experience			X	X	X
Prioritizing personal boundaries	👤 Motivation	Related pursuits		X	X		X
	👤 Topic selection techniques	Based on personal interest		X	X	X	X
	👤 Styling objectives	To cater to their own preferences	X		X	X	
	👤 Styling objectives	To work within personal constraints		X		X	
Novelty and value addition	👤 Motivation	Inadequate current documentation			X	X	
	👤 Topic selection techniques	Filling documentation gap	X	X			X
	👤 Styling objectives	To differ from existing documentation					
Growth and visibility	👤 Motivation	Evangelism and rewards		X	X	X	X
	👤 Topic selection techniques	Based on existing demand	X	X	X		
	👤 Styling objectives	To increase viewership	X		X	X	
	👤 Styling objectives	To cater to learner needs	X			X	X
Content-oriented	👤 Topic selection techniques	Based on the nature of the topic	X	X		X	X
	👤 Styling objectives	To match the nature of the content	X		X	X	

The complete table for all 26 informants is available in our online replication package.

Personal development





A documentor having the *personal development* mindset focuses on how their contributed documentation can be used to improve their own knowledge and opportunities. Documentors contribute documentation for their 👤 *professional development* or to 👤 *capture their learning*, and select topics to cover 👤 *based on their personal experiences*.

Learning through teaching [52, 225] and public note-taking [101, 231] are beneficial for both the creators and the audience. Kim et al.’s theory of learning describes that information gain follows three learning stages: the first has mainly declarative knowledge, the second has a mix of declarative and procedural knowledge, and the third has mainly procedural knowledge [121]. Documentation contribution follows a similar set of stages in which documentors first gather information, try and apply it for themselves, and document that applied knowledge for others to use. In doing so, documentors are able to learn and retain technical knowledge and accompanying soft skills such as communication.

Thus, although the effort required to contribute documentation is high [228], documentors value the experience they gain: “Honestly, I hate writing [...] but I’m writing [the documentation] for myself, because I forget things and if I cannot clearly express myself, it’s going to be very hard for me to understand it later.”[P24] This experience is key for improving the documentor’s own professional qualities: “That ability to summarise, communicate, write it down [...] You have to communicate clearly and well in GitHub issues, comments and reviews. So the fact that I write constantly makes me a better engineer.”[P15] With this mindset, documentors primarily document for themselves, and do so by imagining themselves as the information consumer: “I picture my own days as someone who has limited knowledge about what I’m writing about.”[P13] This system of conversing with the “other self” who consumes and responds

to information that the “self” creates [171] can help provide clear information: “I’m trying to be explicit so that people understand it without getting confused, because I know that I’ll be confused. If I go back to the blog post that I’ve written in the beginning, I hope I can understand what I’ve done there.”[P24]




Prioritizing personal boundaries

As volunteer contributors, documentors have the liberty to mould their documentation to their own constraints and interests, and thus may have the *prioritizing personal boundaries* mindset. This mindset involves creating documentation because of  *related pursuits*, selecting topics  *based on personal interest*, and styling content  *to work within personal constraints* and  *to cater to their own preferences*.

Sansone and Smith defined intrinsic motivation as occurring for individuals when “their behaviour is motivated by the actually, anticipated, or sought experience of interest” [219]. They further described that to pursue long-term goals, individuals must self-regulate their behaviour to continue to be motivated to experience interest, so that they can reach their goals. With the *prioritizing personal boundaries* mindset, documentors pursue topics of their interest, and design their documentation with styles associated with their preferences. As a result, an implicit self-regulation to retain the motivation to contribute documentation exists. Thus, the documentor’s interest is important for documenting. For example, despite the demand for particular topics, documentors can make decisions about what they would like to cover in their documentation: “I just don’t find testing very interesting. So none of my content is about it and I don’t feel like I’m the best person to teach it, as a result.”[P16] This interest can also act as a proxy for what would engage their audience: “If something is really, really interesting for me, that’s a good candidate for what would probably interest all my audience.”[P3]

As voluntary contributors with other commitments, documentors are also bound by the amount of time they have. Documenting familiar content allows documentors to save time because they already have the material they need ready, either in the form of notes (P15, P17, P21) or code: “[The documentation] doesn’t exist until the demo is built. If I do a cool thing and think oh, I should write about that, then [the demo] is already done. Whereas if I wanted to write a blog post about something [I have not done], I would need to figure it out and then write the words. But typically the blogging process starts when I’ve got a demo and all I need are the words.”[P15] To reduce the time spent in actual documentation creation, documentors can allow ideas to first develop in their mind: “As I’m coding, I’m thinking about how my writing should be, I’m thinking about the structure. So when I’m actually writing it doesn’t take much time.”[P7] Other forms of documentation, such as official reference documentation, are continuously revised to be consistent with the corresponding technology [227] to communicate *user-accessible features* [54]. In contrast, documentors do not have the pressure to create documentation: “I acknowledge that [creating documentation] is really time consuming for me. Sometimes I don’t want to do it and then I don’t. So I’m not forcing myself to be active and reach an audience because I don’t want to make this blow up, I don’t want to have a big audience. I just want to teach people that want to learn and I also want to learn.”[P22]

Novelty and value addition





Documentors having the *novelty and value addition* mindset are conscious of the need to provide documentation with a clear value proposition over existing resources. This mindset stems from experience with  *inadequate current documentation*, and involves selecting topics that assist in  *filling the documentation gap*, and styling content  *to differ from existing documentation*.

Existing documentation often has many issues, including lack of readability [7] and irrelevance to real world examples [87], that can hamper the learning of technical concepts [210]. Some of these issues may arise because of the *expert blind spot* [177], wherein experts, who create the technology and the corresponding documentation, make incorrect assumptions about novices' knowledge and understanding when communicating technical details. Instead, users and other participants not at the core of a community can provide novel perspectives and ideas, as opposed to core contributors [218]. The *novelty and value addition* mindset exemplifies the emphasis that documentors, who are not necessarily directly associated with a software technology, put on how documentation can be valuable. Documentors think about how to contribute unique content: "I was looking for ideas and .Net7 has come out, C#11 came out. [...] All the other bloggers are going to pick: *here are the new features in C#11* or they're going to pick topics like *how to write good C*. So I thought, all right, I'm going to choose something totally different." [P6] Documentors identify how to add value to existing documentation: "Sometimes, I will copy a specific example from W3Tutorials or TutorialsPoint because [...] it's a simple example. And I try to explain how I would want it to be explained to me, because that's what they're missing - the explanation." [P18]

As users of documentation, documentors recognize the limitation of existing documentation and have ideas to address the struggles of learning from inadequate documentation. For example, a common documentation issue is the lack of real-world examples in documentation [7, 87]. Documentors who learned software development technology for a specific purpose recognize this issue from their own experiences: "When I was freelancing, I felt that a lot of stuff that I had done earlier on my YouTube channel was theory, but it wasn't real world. And I thought, actually some of this freelance stuff is really real world. And I should make videos on this because these are actual problems and it's not straightforward." [P8] With the important technical knowledge that links the technology with realistic applications, documentors design their documentation to provide novel, practical examples: "I create a small use case that will help put someone in this problem situation, and then I explain that we will build this [solution] [...] At the end, [the learner] will have a project that they can use in the real world." [P23] Ultimately, documentation is a communication [201], and since different people communicate differently, there is an inherent originality to human-created documentation: "There's this whole thing of: the topic has already been written about, so there's no need to write about it. But, individually, we all have various ways we could add value to people that relate to us, by how we frame our writing." [P9] As a result, the value and perspective that documentors contribute can not be overshadowed by emerging generative artificial intelligence (AI) technologies: "When you write *create a list for me*, you get like ten points or twenty points [from ChatGPT]. But when you're putting your own words [...] it's like your own art." [P19] Instead, these systems can be used as tools to support documentors in their creation process [30]: "I think as long as humans are always thinking of ways around problems, we might still just have an advantage over computers and we could just make them do

the boring stuff.”[P8]



Growth and visibility

The *growth and visibility* mindset refers to the pursuit of viewership of the documentation, either to help more people learn, or to reap other rewards such as fame or monetary compensation. Documentors with this mindset are motivated by  *evangelism and rewards*, select topics  *based on existing demand*, and style their documentation  *to increase viewership* and  *to cater to learner needs*.

Kroll introduced three perspectives to thinking about the audience when tailoring written content: *rhetorical*, i.e. the characteristics of an audience, *informational*, i.e. the content they would need, and the *social*, i.e. their attention and preferences [127]. With the *growth and visibility* mindset, analysing what will help engage an audience is part of the documentor’s workflow: “you do research on competition versus volume [for a topic].”[P2] Based on their research, documentors can take informed decisions: “Ultimately what I’m trying to do is help people. So if 20,000 people are viewing a video and 100 people are viewing a blog post, then it makes sense that Youtube is where this content goes.”[P10] Similarly, styling content to capture a potential audience’s attention is critical [160]: “Maybe they [readers] hit ctrl-F and they search for *cluster*. They don’t find it. Boom - we’ve just lost their attention. So catering to how people will search for things, be it text, be it visual, or the specific words they are using [is important].”[P1] Documentors even prioritize such features as their first step of documentation creation: “Titles and thumbnails are very important when it comes to YouTube and click through rate. You could make a great video, but if you are not grabbing someone’s attention with both the title and a thumbnail, it’s kind of pointless. So I always start there.”[P10] Search engine algorithms retrieve development related information from a variety of platforms, including medium.com and youtube.com [96]. As a result, documentors take deliberate measures to ensure their documentation is findable [31] and discoverable [210]: “As much as I have the philosophy of writing for humans first before search engines, you also have to consider how your articles will rank on Google, how your article will be searchable, how your article is easily accessible when people try to query that topic.”[P9]

Documentors leverage multiple platforms for their content, as cross-posting can also increase visibility [157]. For example, P8 developed three formats simultaneously: “The website was something to show people or redirect people to, when I was applying for jobs. And then it was also a way to accompany the videos, so I was hoping that the website would lead people to the YouTube channel and the YouTube channel would lead people to the website. And all the while I was building up my GitHub repositories for my learning and skills.”[P8] Different formats from an individual documentor complement one another: “I just want a blog post [that points to the video], in case someone is searching, to find the video, because that’s where the bulk of the work went into.”[P10] This is especially useful when documentors have paid documentation: “I create the course and then kind of see how I can reuse the content from the course in different ways. So that’s when I actually write a lot of articles, create GitHub repositories and also use the course videos to create a lot of [free] YouTube videos.”[P3] Then, the reuse of information has the ulterior motive of guiding audiences towards the paid material: “I wanted to give people a little taste and be like: hey, if my teaching style suits you, then hopefully you buy the course or follow my stuff.”[P14]

Content-oriented

Documentors having the *content-oriented* mindset allow the environment and circumstances, as well as the content itself, to guide the contribution process. With this mindset, documentors select topics  *based on the nature of the topic* and style their documentation  *to match the nature of the content*.

Prior research has investigated how to present information based on what is to be conveyed to audiences [248] and general best practices to follow [198], without discussing the *applicability* of presentation styles to topics and their information. Wright emphasized how the medium, e.g. printed pages versus dynamic web pages, is dependent on the topic at hand, and the medium should be selected based on its suitability to the topic [264]. Documentors with the *content-oriented* mindset pay attention to what the content is best suited to, especially because there can be different types of knowledge [152] that can be shared. Characteristics of a topic help make documentors' creation decisions: "I actually started with Seaborn for a few couple of months. And then I discovered Plotly which seemed even more intuitive to me, more user friendly and had more options graphing wise than Seaborn and I liked the Plotly Express documentation more, so that's why I moved to Plotly." [P11] Similarly, how to style the documentation also depends on the topics and the content that will go into the documentation: "For instance, I have a tutorial [... like] a set of examples for Numpy whose main theme is drawing stuff in Python. And there is not really an order, especially after you are past the first few chapters. You figure out how to make a picture in Python, and then you can grab whether you need circles first or triangles, or some image filters." [P4] In this case, while the first few sections of the tutorial are sequential to introduce the foundations of drawing in Python, the remaining are modular to allow readers to browse as needed.

It is important for documentors to have thought about the content deeply, even taking a break during the creation process: "It is not okay to come in one day and write the article: draft, edit, and publish. It's really not effective because you have to write and go away from that article, do other things, refresh your mind, and then come back to update the article." [P9] Documentors with the *content-oriented* mindset think about the long-term relevance of particular information, and thus decide whether it is worth documenting: "Once you put the content out there, things change, but they don't change so quickly that it is irrelevant. So it is just really thinking of things as what will be useful for the next year or two to come." [P17]

5.4 Validation

To determine whether the mindsets we elicited resonate with documentors' experiences, we sent a validation questionnaire to our informants. We received 17 responses, and refer to the respondents as R1-R17, in the remainder of the text.^(w) Figure 5.2 shows the responses to the questions with the Likert response format. For each mindset, at least five respondents agreed to having experienced the mindset while contributing documentation. Thus, we can confirm that all the mindsets credibly capture the different thought processes during documentation

^(w)The numbers in these pseudonyms do not correspond to the pseudonyms of the informants P1-P17. We do not make the association between respondents and informants to further protect their anonymity.

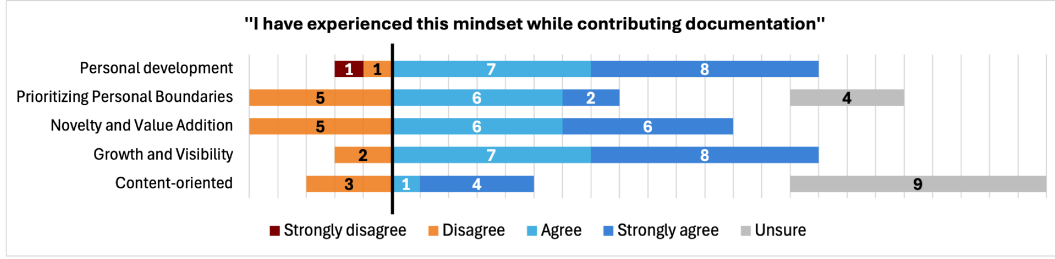


Figure 5.2: Agreement responses of the 17 respondents to the validation questionnaire.

contribution.

We compared the mindsets reported from the questionnaire (perceived) to the mindsets we derived for the same informant (reported) from their interview (i.e. Table 5.5). Although we did not expect to find a perfect correspondence between the two, we noted that there was consistency between the perceived and reported mindsets. For example, R3 “strongly agreed” to experiencing the *growth and visibility* and *content-oriented* mindsets. We note that this was also the case in their interviews — they shared anecdotes for many of the considerations associated with these two mindsets. When a respondent *disagreed* with experiencing a mindset in the validation questionnaire, our mappings showed that the informant did not display that mindset dominantly, i.e. they may have discussed a few of the considerations of the mindset during the interview, but not all the associated considerations. Furthermore, the mindsets capture the *latent* attitude and subsequent thought process. One respondent who selected “strongly agree” to experiencing the *content-oriented* mindset shared: “This is very insightful and I would not have thought of this myself, but I think you are spot on in identifying this as a mindset.”[R1]

In total, we received thirteen “Unsure” responses for the mindsets, with respondents explaining that they did not understand the mindset for four of these cases (one for *prioritizing personal boundaries* and three for the *content-oriented* mindset). We also noted two instances where the mindsets were interpreted differently from our elicitation. For example, R12 “strongly agreed” to the *personal development* mindset and explained: “I focus on the reader”[R12]. However, we associate the focus on the external audience with the *growth and visibility* mindset. These confusions may be because we provided a single-line description of each mindset, which can not sufficiently capture the depth of the mindset. We chose to give only a brief description in the validation questionnaire for two reasons. First, we did not want to overwhelm respondents with too much text that could discourage them from completing the questionnaire. Second, we wanted to understand whether respondents’ experiences resonated with what we had elicited as a mindset. Had we provided an explanation of the considerations associated with a particular mindset, there was a risk that respondents would consider it a hard constraint on what constitutes the mindset. Responses to the open-ended questions validated that respondents understood the general attitude behind the mindset, despite the one-line description. For example, for the *personal development* mindset, R5 explained: “Resonates with Feynman technique. It helps fill the knowledge-gaps while putting the documentation together.”[R5] This is in-line with our elicitation of the mindset.

5.4.1 Study Design Trade-offs

Our informant pool is a convenience sample of identified documentors whose contact information was publicly available. The limited number of informants is a consequence of conducting lengthy interviews. There is a possibility that other considerations and mindsets exist for documentation contribution. We chose to accept the trade-off [209] of fewer participants from interviews, as opposed to potentially more participants from a survey study, in favor of obtaining deeper insights on the documentation process. Any novel observations of the documentation process in future work can be integrated into the proposed framework in Figure 5.1.

The interview methodology relies on reflection and self-reporting, wherein informants must recall and describe their own experiences. There is a risk of differences between perceived and actual documentation contribution behaviour. An alternative would have been to conduct an observational study in a natural or lab setting. However, in a natural setting, observing and analysing the documentation process is not feasible, as documentation contribution does not necessarily have a set time frame. Alternatively, developing a simulated setting for a lab study would remove the important context of *volunteered* contributed documentation. Thus, we decided to conduct interviews. We followed a specific-to-general interview technique [99] in which we asked informants to explain their thoughts and actions with concrete examples and then followed up with a question on whether this was their general procedure. This technique helped provide context and evidence for the interview responses.

5.5 Implications

The responses to the open-ended questions in our validation questionnaire complemented the interviews to provide further insight about respondents' creation processes. We reflect upon how the mindsets are at play during documentation contribution, and use quotes from the interviews and validation questionnaire for insight. Specifically, we note how documentors must maintain a balance between multiple mindsets, the challenges they face in pursuing particular considerations, and mindsets that respondents suggested, and ideas for future work related to these aspects.

5.5.1 Balancing Multiple Mindsets

The five mindsets we elicited are neither exhaustive nor mutually exclusive, and documentors may display more than one mindset in the creation process: "As a solo content creator, I had to experience most of the mindsets at once, since they all played a factor for me." [R7] In fact, having a single mindset can even be harmful to the quality of the documentation produced. For example, while being *content-oriented* is beneficial to ensure that the style, topics, and information content go hand-in-hand, it is easy to miss what users may prefer.

Documentors must also think about the visibility of their documentation: "We dedicate a bunch of pages to [a topic] and [the documentation pages] don't get used. It's essentially really thinking about our

users in terms of what they might want to do and just having examples for that.”[P1] However, focusing only features that promote visibility via search engine optimization (SEO) can interfere with the design of documentation: “Sometimes putting an image [in the documentation] was not really necessary, but I have to, in order to optimize for SEO.”[P23] Such overhead can also disrupt the learning and creation workflow: “[...] sometimes it bores me to create a new post [...] doing the alt tags, putting an image, cropping it, making sure that the key phrase is in there, the SEO stuff. It might take an hour. You know, in that hour I could have learned that [topic] in five minutes and then in the other 55 minutes I could have gone off and learned something way more advanced.”[P8]

Thus, maintaining a fine balance between the mindsets is an integral part of the documentation contribution process. Depending on the goals of the documentor, each mindset may not be weighted equally. For example, R4 and R11 agreed to experiencing the *growth and visibility* mindset, specifying the caveat: “but it is not my top priority”[R4]. The mindsets may also be prioritized differently, based on the documentor’s own environment and experiences. One respondent disagreed to having experienced the *novelty and value addition* mindset: “Being too creative with the documentation design could take extra effort, and there is usually no value in doing so.”[R5] Another respondent who disagreed to experiencing the mindset explained that originality occurs naturally: “There might be overlap, but every time someone else is writing about the same thing, it will come out differently.”[R12] In contrast, respondents who agreed to having experienced the mindset emphasized how important it was to intentionally be different: “So much programming content is boring and hard to digest (for me and many of my previous classmates). I knew there was a gap on YouTube for it, and I could fill it.”[R7] We asked respondents to answer the questions based on their own experiences, however, documentors may also recognize mindsets in other documentors. For example, one respondent selected “Unsure” for the mindset *prioritizing personal boundaries*, and explained that “I have not experienced this mindset, but I can see how others might adopt it in their work.”[R1].

Ultimately, “Embracing a combination of these mindsets can lead to the creation of comprehensive, valuable, and user-friendly documentation that encourages documentors to stay up-to-date with the latest developments in their field and incorporate new knowledge into the documentation.”[R16] As a result, further research can investigate how to support documentation creators in balancing and managing multiple mindsets to produce high quality and relevant software documentation.

5.5.2 Challenges with Pursuing Considerations

With the freedom to prioritize considerations as they like, documentors must consider the trade-offs of their decisions. For example, “Documenting a fresh or rarely seen content [...] Brings less traction but feels more rewarding, especially because that small audience appreciates those kinds of contents most of the time.”[R11] Furthermore, documentors can become overwhelmed because they must also handle the many tasks related to curating the documentation content, such as editing and audience engagement.


For documentors of video tutorials, creating the video takes additional effort: “I also do all of my own editing and planning. So you can get really lonely. That’s what people don’t really talk about is that, on YouTube, you’re all of these jobs in one. Which is, you know, it’s a lot of work.”[P18] Documentors must then *prioritize their personal boundaries* and use multiple strategies to optimize documentation

creation. For example, when documenting topics that they already have experience with, the material is ready in advance: “[...] it’s something that I’ve tried and practiced myself and went back and forth, tried something, it failed, tried something else; I keep notes of all that process.”[P21] Similarly, planning by thinking over the content, developing a structure, and writing the code first can help speed up the creation: “Before I write anything, I think: this is how I’ll do it. [...] it’s kind of stuck in my head for a while and actually writing is the fastest thing.”[P24]

Identifying and connecting with the audience is an integral part of communicating documentation [32, 261], and is related to the *growth and visibility* mindset. However, documentors do not have a direct way to communicate with the target audience and understand their actual needs: “I wish I understood better how people are discovering my blog and what they are hoping to get from it.”[P16] They can interact with their audience via comments on the documentation, or through social media, but this becomes impractical with greater visibility: “there’s just so many comments that I’m not going to respond to everyone.”[P17] When audience’s needs are difficult to gather, documentors must rely on the *imagined audience* [63, 145] and online metrics such as popularity trends of technical topics. Further work can investigate how to support documentors in pursuing relevant considerations towards achieving their goals. For example, prior work in the domain of social media has shown that the duration of videos impacts the level of user engagement [2]. Similarly, objective metrics such as page visits or video views can be used to assess the extent to which selecting topics *based on existing demand* can help achieve the *growth and visibility* mindset.


Still, “You can’t really control who is going to read your docs and what they’re going to need. You have to look at all of the things that people might need to know and make some arbitrary cutoff decisions.”[P1] Such arbitrary decisions may impact *completeness* [7], i.e. how well the documentation informs about a software [239], as not all areas of a technology may be covered [188]. However, documentors, as volunteer documentation contributors, may have mindsets that do not focus on contributing *complete* documentation. Ultimately, documentation is “[...] a long term project. You really have to be blogging for years before anyone starts paying attention.”[P16] With all the time, effort, and thought spent in contributing documentation, documentors must be cautious of their expectations of return: “you have to try to enjoy the process, enjoy learning new things and trust that it’s a long game. It’s not - get rich or get famous overnight - type of thing.”[P10]

5.5.3 Other Mindsets

Eight respondents answered the question “Were there any other mindsets that you experienced that we did not ask about?”. In five cases, the responses are subsumed by our elicited mindsets. For example, R1 and R4 referred to the open source software (OSS) community, where the idea is to share knowledge to “give back” and “empower a greater community”: “Personally, I feel my career and education have benefited a lot from freely accessible content online. Part of the reason I create content (and similar things like contributing to OSS) is to, more or less, return the favor.”[R1] We capture this under  *evangelism and rewards*, which is a consideration associated with the *growth and visibility* mindset.

R5 described the important role of source code in informing the documentation: “Writing documentation usually makes the author revisit the source code. This might give them ideas about refactoring,

or getting a better view of things that should be done, leading to writing code that is more clear, removing bad logic, handling edge-cases, etc.”[R5] R16 also described that the feedback loop “involves regularly revisiting and updating the content based on user feedback, new insights, and changes in the underlying technology or process.”[R16] We suggest further investigation to understand this cycle of code and documentation. Other project stakeholders also influence documentation creation. For example, documenting for colleagues (R13) rather than end users, is helpful to orient teams as well as provide clear insight to businesses about the value of the software (R5). These responses suggest that the context of collaborative software creation groups and businesses may introduce new goals, for which our framework of mindsets and considerations can be applied and expanded.

R7 suggested the “empathy mindset”: “Being a college student struggling with Computer Science, feeling that frustration, pain and confusion first-hand was something I knew many others were facing [...] I teach them how I would have wanted programming to be taught to me.”[R7] This respondent captures an integral aspect of the mindsets and the considerations: the *emotions* of the documentor.  *Inadequate current documentation* captures the experience of finding a gap in available information that motivates documentation contribution. However, it does not describe the frustration and struggle of experiencing this motivation, that can impact documentation creation. While our focus is on why and how documentors think, future work can explore the emotions of software documentors that contextualize the thought process, and how they impact documentation contribution.

Chapter 6

Interactions with Multimodal Documentation

Our insights from Chapters 3 to 5 indicate that current techniques to consider varied user preferences for software tutorials involves managing and maintaining documentation of different formats across different platforms. Furthermore users have different preferences about documentation, which can depend on their roles and responsibilities [62]. However, prior work has reported that documentation can suffer from *incompleteness*, wherein the documentation is missing important elements such as step-by-step guides, code examples, and code comments [7] that can cater to varied needs. In this phase, we investigated how a single document can cater to the varied presentation needs and preferences of users for different types of programming tasks, through information presented via multiple *modalities*. We introduce the term *modality* to refer to a presentation format, such as text content or a table, that is used to present information within a documentation resource.

We created three multimodal tutorials prototypes about three basic programming concepts in the Java language, namely regular expressions, inheritance, and exception handling. Whereas it may seem intuitive that users' modality preferences can vary based on the specific types of tasks they perform, we found little evidence in the literature to support this hypothesis. To test this hypothesis, 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.

Goal

The goal of this phase of the research was to understand how programmers interact with multimodal documentation and how the different modalities could serve different users' needs and preferences.

Research Question

How do programmers make decisions about their presentation needs and preferences in a programming tutorial?

Publication

The study of programmers' decision making when accessing multimodal software documentation was published in the article *How Programmers Interact with Multimodal Software Documentation* [23].

Replication Package

The documents needed to replicate the study are available in Appendix D.

6.1 Study Design

We created a prototype multimodal tutorial for each of three Java programming topics. Although a number of documentation types exist [198], we focused on *tutorials*, as they are a commonly used source of software technology information [96]. We conducted a survey to understand whether programmers found the modalities useful and how they used them to complete three types of programming tasks.

6.1.1 Multimodal Tutorial Prototype

We developed a prototype tutorial as a static HTML file supported by Javascript and CSS. In the multimodal tutorial, 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 6.1 shows an illustration of part of a multimodal tutorial. The five modalities (represented by ■), as well as the additional features (represented by ●), are as follows:

■ Text content: Text-based tutorials are the most common source of programming information [69]. 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* [168].

■ Regular code examples: Code examples can contribute to a document's effectiveness [73]. Developers search for code snippets, especially to reuse the snippet for their own use cases [268]. We provided information in regular code examples, i.e. complete code snippets with no additional comments or annotations (see Figure 6.2a).

■ 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 [210].

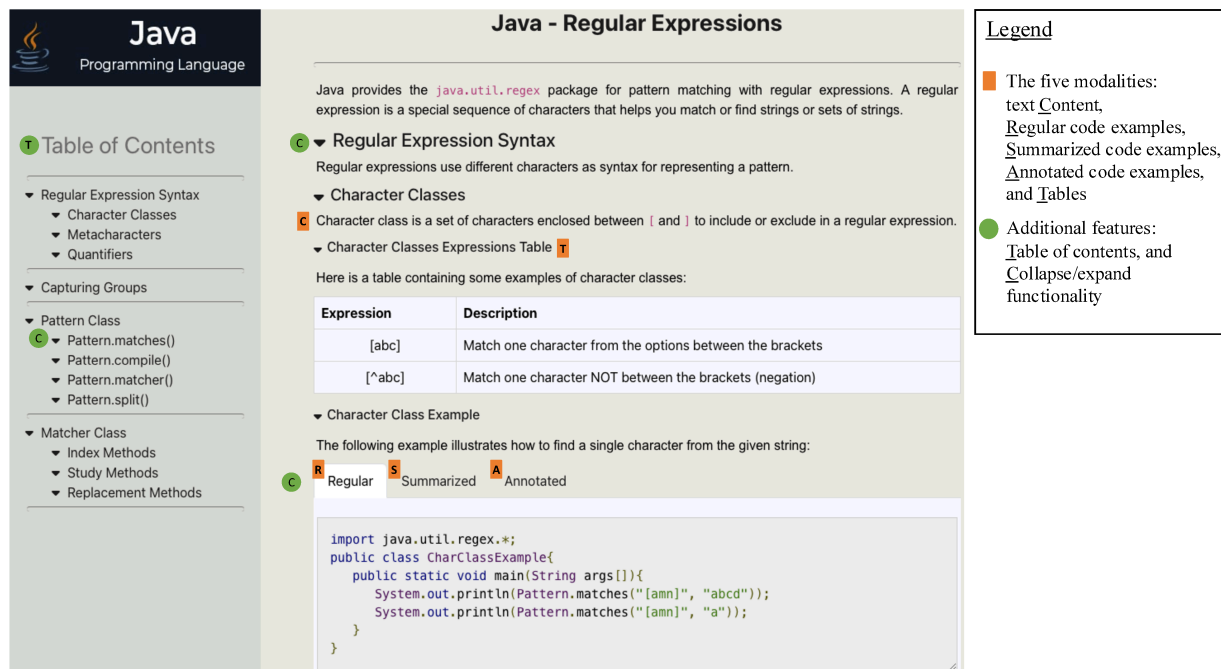


Figure 6.1: Illustration (with excerpts) of a multimodal tutorial for regular expressions in Java. The tutorial prototypes we created for each of the three topics, i.e. regular expressions, inheritance, and exception handling, provide more information through each of the modalities.

We provided *summarized* code examples (see Figure 6.2b), by following the selection and presentation practices for summarizing code examples reported by Ying and Robillard [274].

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 [174] (see Figure 6.2c). We refer to them as *annotated* code examples. We provided the three types of code examples through navigation tabs as shown in Figure 6.1, 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 [104]. 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 [176, 260], 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



Figure 6.2: The three code example modalities to demonstrate how to implement the character classes in regular expressions for Java.

navigating through the main tutorial body [24].

• **Collapse/expand:** Prior work has suggested that users can benefit from having information revealed to them gradually [176]. 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 including `beginnersbook.com`, `javatpoint.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*.

6.1.2 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 on the 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

Table 6.1: 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 <code>matches()</code> method in the <code>Pattern</code> class or the <code>matches()</code> method in the <code>Matcher</code> class. What is the difference between both these methods?
How-to	The user is asked to input their email address in the expected format: <code>username@domain.com</code> . 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	<p>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 <code>matches()</code> method:</p> <pre>\d{3}-?\d{3}-?\d{4}</pre> <p>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?</p>

tutorial page, and asked participants to complete three programming tasks. The three programming tasks are based on *search intent* categories proposed by Rao et al [204]. 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 *task types*, i.e. three types of programming tasks, namely *conceptual* (corresponding to the *Learn* intent), *how-to*, and *debugging*.^(x) Table 6.1 provides an example of each task type. We created one programming task corresponding to each of these task types, for each of the three programming 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. The complete list of tasks are available in Appendix D.1. 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 6.3). In the survey, we referred to the modalities also as “features” to have a relatable terminology for respondents.

In the fourth and 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, as shown in Figure 6.4.

^(x)We chose to rename *Learn* to *Conceptual* in our study, because all task types may involve some learning. Instead, conceptual specifically refers to learning about a topic, such as comparing two concepts.

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	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Regular code example	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Summarized code example	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Annotated code example	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
Table	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

19

Please explain your choices in Question 18.

★ 📄

Enter your answer

Figure 6.3: 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 6.1.2).

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.

6.1.3 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

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.

Figure 6.4: Optional open-ended questions in the survey.

Table 6.2: Demographics of survey respondents.

	#Res.	Age					Gender			Region			P.Exp. (yrs)		
		18-24	25-34	35-44	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

N.S. — Prefer not to say

P.Exp. (yrs) — Programming Experience in years,

N.A. — North America

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.

6.1.4 Analysis

Table 6.2 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 [164] to account for the multiple categories of each dimension. We also applied a Bonferroni correction by multiplying the p-value obtained from each test by 16, to mitigate Type-I error when making multiple comparisons [3]. 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 6.3.

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 [224]. 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

CHAPTER 6. INTERACTIONS WITH MULTIMODAL DOCUMENTATION

Table 6.3: 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.

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 6.2. Appendix D.2 and Appendix D.3 contain the statistical analysis results not discussed in Section 6.2.

6.1.5 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. [209]. 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 6.2.4).

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 6.2.3). This may be because the question required some inference from the tutorial content, which could have been easy to miss.

6.2 Programmer Interactions 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.

6.2.1 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 6.5). This is also visible from the residuals which show that the text content being *very useful* is observed more than expected (see Figure 6.6a). 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"[11].

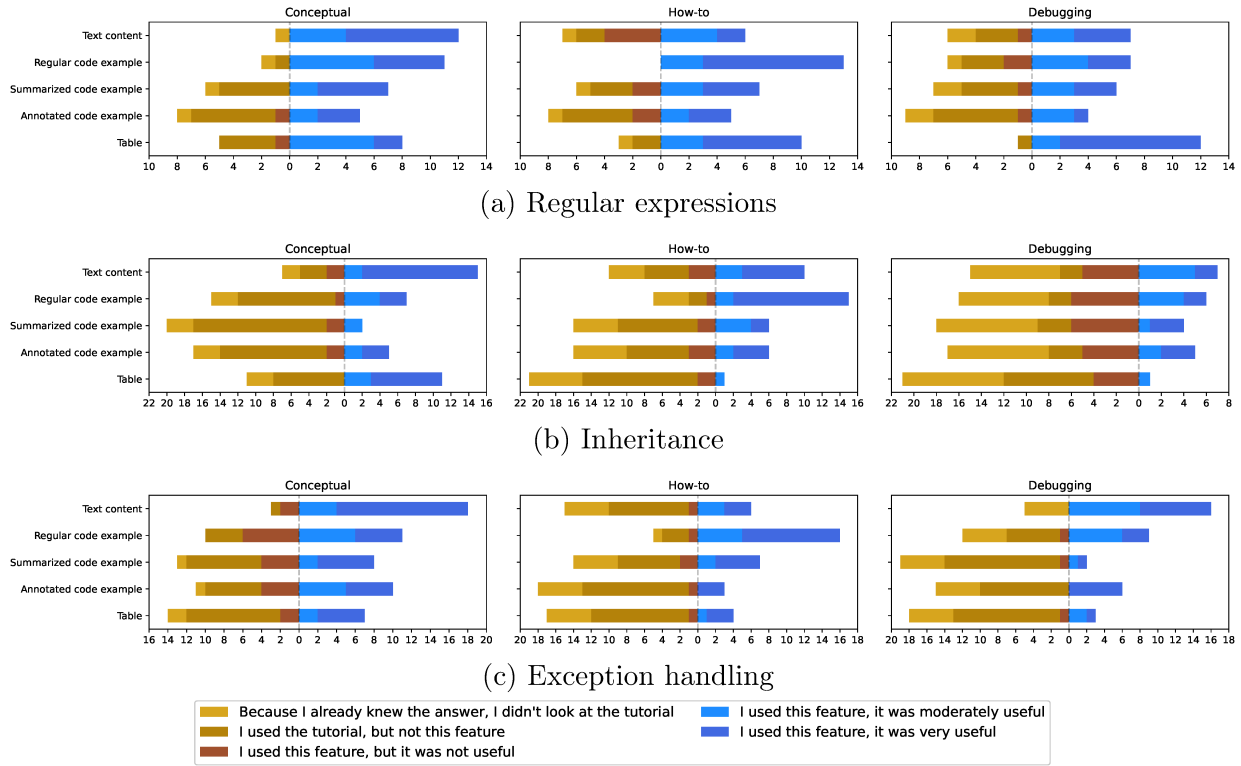


Figure 6.5: Rating of usefulness for the five modalities, per task type and topic, for the three multimodal tutorials. Note that the legend refers to modalities as “features” (see Section 6.1.2).

For the conceptual tasks, respondents found regular code examples the next most useful, after text content, for regular expressions and exception handling (see Figure 6.5): “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 6.6a).

For inheritance, the tables were the second-most useful modality (see Figure 6.5). 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.

		Rating				
		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
Modality	Text content	3	4	4	10	35
	Regular code example	4	16	7	16	13
	Summarized code example	5	28	6	6	11
	Annotated code example	5	24	7	9	11
	Table	5	22	3	11	15

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

		Rating				
		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
Modality	Text content	10	16	8	10	12
	Regular code example	5	5	2	10	34
	Summarized code example	11	19	6	9	11
	Annotated code example	12	24	6	4	10
	Table	12	26	3	5	10

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

		Rating				
		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
Topic	Regex	1	7	1	11	19
	Inheritance	18	29	6	5	8
	Exceptions	12	33	4	5	9

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



Figure 6.6: 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 6.1.2).

6.2.2 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 6.5). The residual for regular code examples being *very useful* for how-to tasks indicate a larger frequency than expected by chance (see Figure 6.6b). 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.

6.2.3 Modality Ratings for Debugging Tasks

For debugging tasks, there is no particular modality which was most useful across all three programming topics (see Figure 6.5), 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 6.5 and 6.6c 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 23 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 6.5). Furthermore, 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]

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

6.2.4 Usefulness of Individual Modalities

Our results from Sections 6.2.1 to 6.2.3 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.

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 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.

6.2.5 Usefulness of Additional Tutorial Features

We report on the two optional open-ended 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] Additionally, 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 especially the case 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]

6.2.6 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 additionally 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]

Chapter 7

Discussion

Our four studies on software documentation reveal that designing documentation involves more than simply providing information about a software technology. Information content is only one characteristic of documentation resources, and is not the only aspect programmers use to identify relevant and useful software documentation. Instead, documentation acts as a *communication* [201] between documentors and programmers. For this communication to be successful, documentors must be able to understand not only the information needs, but also the structure, organization, and presentation preferences of their audience. Furthermore, as users of software technologies, documentors may incorporate their own preferences to inform the documentation they create such that it can be relevant to other programmers in similar positions.

Our findings suggest the need for customizable documentation. We discuss important insights from the four phases of our research about software documentation. We also discuss the anticipated challenges of building such versatile documentation given these insights. We discuss avenues for future work towards the vision of customizable, multimodal software documentation that can cater to various needs and preferences while retaining documentors' considerations. Henceforth, we use the term *documentors* to refer to documentation contributors, and the term *programmers* for people who look for and consume information about a software technology.

7.1 The Software Documentation Environment

Prior work has primarily approached software documentation as a source of technology information. However, our work provides insights on software documentation that allow us to understand its value and context through the perspectives of people who interact with it. Specifically, we note that *software documentation is human-centric*. We discuss the *management of multiple documentation types* and insights on the *design of customizable documentation*.

7.1.1 Software Documentation is Human-centric

Although the primary focus of software documentation may be to provide *information* about a technology, the actual documentation process from creation to use is influenced by the *humans* who interact with documentation.

Documentors play an important role in creating documentation; they are responsible for designing documentation to help programmers learn about a software technology. Whereas developers consider documentation as a product that is a supplementary part of a software package [205], time and effort goes into designing and creating documentation. Since documentors can make decisions about the documentation they contribute, they can prioritize their own needs, interests, and personal development. Whereas existing software to support the creation of software focuses on the content of documentation [25, 91], documentation creation tools can better support the creation and contribution process by considering the needs and preferences of the documentor.

Once produced, the effectiveness of documentation depends on the needs and preferences of the programmers who access it. Additionally, documentation can only be useful if it can be successfully located by programmers [7]. Moreover, documentation that is not organized or presented in a manner that a programmer prefers, also impacts their information consumption. For example, boilerplate documentation that simply restates information that can be gathered by a method’s definition is not useful, and thus frustrating to programmers [210]. Furthermore, as reported in Chapter 3, if documentation is not styled in a way that the programmer prefers, it may result in abandonment of the resource, despite it containing valuable information. Instead, software documentation itself must be designed to meet the needs of the intended audience [211].

Naturally, documentation creators who are themselves users of software technology documentation, may have some insight on documentation use. This forms a connection between documentors and programmers. Thus, as consumers of other documentation, documentors are well-informed to cater to the needs of similarly positioned information seekers. Prior work has discussed documentation creators as evangelists [156]. We introduce a new perspective: volunteer documentation contributors as documentation consumers. Documentors leverage their own learning and programming experiences to inform decisions during documentation creation. As a result, they are informed of the needs of users, an important feature of software documentation production [144]. Additionally, as the preferred style of learning influences the documentation a documentor creates [57], the documentation created can serve audiences who have similar preferences [62]. This indirect interaction is a partial view of Mehlenbacher’s predicted input-output model of documentation creation that suggests that technology creators, documentation writers, and end users are a “triangle of interrelated technology users” [163].

With *documentors* in control of how documentation is designed, and *programmers* in control of how they search for information in documentation, there is a need to incorporate the considerations of both of these stakeholders. However, our findings show that each group has variations among their needs, preferences, and priorities. Furthermore, a direct communication between documentors and programmers becomes a practical scalability issue

with increased documentation visibility. As a result, there is a need for convergence and communication between documentation creators and user-programmers.

7.1.2 Management of Multiple Documentation Types

There are a number of types of software documentation [96], such as API reference documentation, software tutorials, textual blogs, YouTube videos, etc. Our findings indicate that different formats for presenting information serve different purposes for both documentors and programmers. For example, we noted from our first and fourth studies that programmers preferred textual explanations to answer conceptual questions, and used code examples to refer to a practical demonstration of these concepts. For documentors in our third study phase (Chapter 5), text-based documentation types such as tutorial blogs acted as a navigation cue for programmers to be redirected to the documentors’ main content, such as Youtube videos or paid course material.

Despite no enforced standard for how different types of documentation should be designed, frameworks such as Procida-Diataxis [198] help provide guidelines for information content. Additionally, we found that both documentors and programmers leverage their prior experiences and familiarity with documentation to identify what kinds of documentation resources contain what kind of material. Whereas programmers use implicit *cues* to navigate, documentors report that some information content naturally lends itself to a particular format. For example, text or videos cater to lengthy conceptual explanations, but overviews are best suited to be presented as bullet points or within tables.

Our studies suggest that presenting information through different modalities, and through different documentation styles can be beneficial. However, it does mean that the same technical information is duplicated across different resources [61]. Consequently, *manually* managing this duplicate information across different resources can lead to inconsistency [15]. There is also an increased effort for documentors who must design each resource such that it is self-contained, while maintaining the correspondence between resources. Furthermore, variations of documentation add to the already numerous resources that programmers must wade through to find pertinent information. Thus, there is a need to support the management of information consistency across multiple documentation types.

Prior work has investigated methods to automatically generate documentation from code comments [91, 125], present information in an alternate manner for code snippets [176], and support the decision-making judgement of reusing knowledge [146]. In the domain of content creation for social media, challenges of managing multiple platforms are similar to information management across multiple documentation types. There, prior work has suggested that the “creator ecology” should be configured to provide creators the support to manage multiple social media platforms, such that they can focus on the actual creation of content [151]. Our observations indicate the need for a framework to repurpose existing content to maintain traceable links between the different software documentation resources. Such a framework can mitigate the risks of information misalignment or inconsistency [202]. Additionally, we propose a movement towards multimodal, customizable documentation that contains information through multiple modalities within a single resource. Whereas the goal

of such design is to cater to variations in design preferences, it would also reduce the effort required to cleanly maintain resources across different formats and platforms.

7.1.3 Design of Customizable Documentation

Designing customizable documentation introduces multiple aspects that must be taken into account. We discuss the aspects that surface from our research.

Incorporating multiple modalities

Software Engineering practitioners have preferences for different types of *genres* or information presentation formats, including tutorials, code or data examples, and tech notes [62]. A documentation such as a tutorial can have multiple elements such as overview information, code snippets, and advanced pages [260] in addition to sections, links, and images [244]. However, programmers find it difficult to navigate documentation and identify the location of the information they are looking for, i.e. in Concepts, API reference, Samples, or in another section [165]. Instead, a customizable documentation should allow programmers to retrieve the same information from any *modality* of their choosing.

Conciseness of information

Programmers may prefer content to be short and concise or, alternatively, detailed. For example, some developers refer to code in order to duplicate it [124], and thus may prefer *complete* code examples that can be executed [24]. Other developers have described that small code examples that show patterns of method usage are more useful than an example of a single call to the method [210]. Finding a common 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 [7]. Instead, providing content in both a concise and detailed format will allow users to make a decision about what amount of information is best suited to their needs. This design can be implemented by allowing users to select between seeing overview or detailed text, complete or summarized code examples, and focused or elaborated tables.

Content visibility configuration

Prior work has suggested that programmers can benefit from having information revealed to them gradually [176]. Following the design guidelines of having multiple formats and variations in the conciseness of content, the ability to show and hide irrelevant content is integral. For example, in the case of overview and detailed text, a user should be able to hide detailed information, should they only want to see an overview. Similarly, sections can be divided based on *core*, *nonfunctional*, and *peripheral* topics [24], allowing programmers to customize the resource to view only core topics if their goal is to be introduced to the software technology. Such control can allow a multimodal documentation to reproduce familiar documentation types. For example, for quick referencing, a programmer can collapse

all other modalities except tables to emulate the typical presentation style of API reference documentation.

Supporting documentation navigation

Organizing a resource into sections and subsections are integral attributes for users [73]. Providing information so as to allow both sequential and modular navigation through a documentation resource [24] involves creating sections that can be navigated to directly. Additionally, cues that programmers use to identify and navigate amongst resources can differ based on information search context. For example, cue-following behaviour can differ based on whether developers are looking to learn about a software bug or fix it [193]. Consequently, tools to support cue-following behaviour should be context-specific: cues that point to code execution and output can be useful for learning, whereas cues to meaningful annotations of relevant source code can be useful for fixing bugs [193]. Thus, presenting explicit cues that programmers can follow can support the efficient search for information.

Making customization changes visible

In a customizable documentation, programmers should be able to manipulate the design of the resource. However, if during customization, changes in the resource are very subtle, then users may have the impression that their efforts to customize the resource make no difference [5]. Thus, it is important that programmers are able to see visible differences in the documentation upon configuring it to their needs. When giving programmers control to manipulate documentation design, changes can be applied immediately, such as changes to settings of color shown immediately [139]. Alternatively, configuration options can be selected and accumulated, and then applied all together allowing a user to directly see a new documentation build, at once [199].

7.2 Anticipated Challenges to Designing Customizable Documentation

Our research shows the potential of multimodal, customizable, versatile documentation. However, based on our findings, we anticipate challenges in designing such documentation. We discuss the challenges related to the *shift of design effort from documentor to programmer*, the *evolving design needs and preferences*, the *evaluation of software documentation quality*, as well as the *impact of artificial intelligence on documentation creation and use*.

7.2.1 Shift of Design Effort from Documentor to Programmer

Caponi et al. proposed a framework for template documents to support the creation of documentation [42]. Whereas their framework is useful for the creation of subsequent documents, our findings suggest that a similar framework that allows users to customize documents through a WYISWYG (What You See Is What You Get) editor, can be useful. However, the

vision of customizable documentation that allows users to manipulate a resource to provide the look-and-feel that would cater to their needs and preferences shifts some considerations of documentation design from the documentor to the programmer seeking information. In principle, this shift of effort should allow documentors to focus on the information content, rather than in developing robust structure and presentation aspects of documentation resources. In a study evaluating a user-controlled text summarization technology, participants described that changes in the text were subtle, giving the impression that their interactions with the technology did not make a difference on the text [5]. Thus, documentors will be faced with an additional challenge in providing user control, i.e. making a resource clearly and visibly manipulatable. Furthermore, documentors will need to make decisions about the extent of control provided to programmers to customize the documentation. For example, whether users should be able to manipulate the smallest units of change possible such as the font size of the text, or only larger structure-related changes, such as the order of sections.

Additionally, a system in which the programmers are given the responsibility of manipulating documentation assumes that they are aware about their needs and preferences. However, prior work has shown that this is not necessarily the case, and often programmers struggle to articulate their needs clearly [149]. Additionally, programmers will have to customize a resource to their needs and preferences *before* being able to consume the documentation resource, and will thus be burdened with additional effort. Already, our results from Chapter 3 show that finding pertinent documentation is an involved process. Programmers may then find the process of searching for information within a pertinent resource fatiguing if they have to first customize it to their needs.

An alternate approach to considering this shift of control effort could be to develop a programmer search profile that could be used to automatically configure documentation to the programmer’s needs and preferences. However, the development of such a profile could pose a threat to the programmer’s privacy [211]. Additionally, our findings from the fourth phase of our research indicate that the use of modalities can also vary between programming task types. Even with a developed user profile, it would be challenging to identify the task at hand or the intention of the programmer when accessing the documentation, without further information on the context of information search.

7.2.2 Evolving Design Needs and Preferences

We find that the needs, motivations, and preferences of documentors and programmers are subject to variation, based on their personal experiences and the context of their creation or search environment, respectively. Similar to a software project with changing requirements [68], the considerations of documentors and programmers can also evolve with time. For example, one documentor from our third study described that they began to contribute documentation many years prior for their *professional development*. But now, their motivation was to earn income from the content they produced, and thus their motivation shifted over time to *evangelism and rewards*. Programmers’ level of background knowledge can impact their information search techniques and needs. Thus through the learning process, and familiarity with documentation resources, programmers could change their search tactics.

For example, we noted in Chapter 3 that programmers used IMPRESSION FACTORS to develop BELIEFS and specify PREFERENCES that impacted future searches. In our fourth study (Chapter 6), respondents referred to particular modalities that could help them recall any prior knowledge they had, e.g. code examples if they had familiarity with coding and just needed to refresh their memory on syntax, or text content if they wanted to confirm their prior understanding of a technical concept.

With evolving software changes, traditional forms of documentation, such as API reference documentation, are continuously revised to be consistent with the corresponding technology [227]. This is especially the case for *official* documentation, i.e. when the documentation is created by technology creators themselves, as they need to communicate any new *user-accessible features* [54] such that the technology can be used effectively. However, revising documentation based on evolving design needs and preferences is non-trivial. This is because any changes can result in additional design variations, with human-creativity at reins. Furthermore, identifying context poses a major challenge, let alone an evolving context. Prior work has studied how people establish context while programming, as well as the activities that programmers perform in the process [47]. Prior work has also attempted to support information search by maintaining track of code changes in the local development environment and identifying web pages which may have relevant information [214]. Still, determining the thought processes of programmers and documentors and the appropriate resource styles that could fulfil their needs remains an open challenge.

7.2.3 Evaluation of Software Documentation Quality

Establishing a common definition for “software documentation” has been a challenge in software engineering research. Instead, prior research has relied on defining the scope of software documentation for their particular study. For example, Forward and Lethbridge, who studied the relevance of software documentation referred to it as “an artifact whose purpose is to communicate information about the software system to which it belongs, to individuals involved in the production of that software” [73]. Ellman studied similarities in documents “within a software documentation” and scoped their data thus: “we are studying software development documentations that were created and/or maintained by software communities active in Stack Overflow, in Eclipse Bugzilla and on YouTube” [64]. In our research too, we found that what constitutes software documentation varies based on peoples’ interpretations. For example, when recruiting documentors for our study in Chapter 5, multiple potential informants expressed that they did not believe they were contributing documentation, but rather were providing technical information about a programming language through videos or blog posts. However, with the surge of platforms to host information [246], previous clear boundaries of software documentation, such as installation documents and reference manuals [232] have blurred to incorporate “non-traditional” sources of information, such as Stack Overflow and blogging platforms [201].

In our study reported in Chapter 4, we focused on only one type of software documentation, namely software tutorials. We initiated the study with the goal of defining what a tutorial was. However, we noted that the variations in tutorial design did not support

the development of a one-size-fits-all definition of a tutorial that could capture its “typical” design and information content. Instead, we were able to develop a framework that could be used to characterize tutorials based on their design-related properties.

Prior work has investigated the *quality* of software documentation using objective metrics that relate to the existence of components [259], measures of readability, and coverage of information [277]. However, we find that the important aspect of human-involvement in the contribution and use of documentation introduces subjective qualities. For example, for documentation contribution, an important consideration is the documentor’s constraints of time and interest. As a result, common standards of documentation design [106, 198] and metrics of quality, such as completeness of coverage of information about a technology [7, 222, 239], may not apply to *voluntarily contributed* documentation. Additionally, previously undiscussed challenges arise with contributed documentation. The pursuit of visibility introduces the risk of prioritizing the design of documentation for search engines over human audiences. In the case of documentation use, the context of a programmer’s information search, such as the task they need to complete, plays a role. Thus, the quality of an information-encompassing customizable documentation would have to consider how the documentation was modified for use. Consequently, although it is necessary to have measures of quality to mitigate difficulties in adopting software technologies, our results point towards the need of an adapted framework for evaluating documentation, that focuses on the considerations of documentors, the design and information preferences of programmers, as well as the information-seeking context.

7.2.4 Impact of Artificial Intelligence on Documentation Creation and Use

In the present world, the application of “artificial intelligence” (AI) is ubiquitous. Large language models (LLMs) are able to generate cohesive text that may be mistaken for human-generated text. Such is the case, that there now exists research on how to differentiate between LLM-generated and human-created text [240]. Additionally, there has been an increase in the number of automated writing assistants powered by AI, such as CodeX [48]. Researchers have explored the potential of using such AI tools in the domain of software engineering [98], for example to support code understanding [173] and generate software documentation [236, 243, 256]. The surge of intelligent writing tools especially, has introduced a worry about whether humans would have to be involved in any form of writing again, including the creation of software documentation.

However, from our third study (Chapter 5), when asked, most documentors did not indicate a concern about their contributions being threatened by AI-powered tools. There still exist issues with LLMs generating inaccurate information in the context of software engineering [70]. Additionally, current AI tools cannot generate the unique “voice” [212], i.e. the experience-backed perspective, that the documentor brings to their documentation. Instead, such tools can take one format that a creator creates and repackage it as required to other formats, thus saving creators from effort in managing multiple formats. Still, AI-based tools require design changes to truly support a documentor’s creation process [30]. Documenta-

tion creation tools should have good support for the *communication* of knowledge [73], which can be subject to the ideas and goals of different documentation creators. However, existing AI-based tools are currently not mature enough to contribute the creativity and uniqueness that result from the *novelty and value addition* mindset and the resulting originality that comes across in the documentation. Thus, in terms of documentation creation, there is still a critical need for the human-in-the-loop.

In terms of documentation use, algorithm-backed conversational agents, such as ChatGPT [180] have made access to information “easier” wherein users can use prompts to retrieve the information they need. Such tools bypass the need to manually search through multiple documentation resources, and instead act as a source of information. Recent work has studied the way in which software engineers interact with ChatGPT and reported that they use it to learn about a technical topic or solve programming tasks [120]. Xiao et al. contributed a dataset of interactions between software developers and ChatGPT [269]. Furthermore, they suggested a number of potential research questions that can be explored, including “How does the code generated by ChatGPT for a given query compare to code that could be found for the same query on the internet (e.g., on Stack Overflow)?”.

Prior work has shown that the responses of ChatGPT can be inconsistent and misleading, which is a concern for learning environments [272]. However, even if AI-backed conversational agents are able to provide accurate information, they do not currently account for structure and organization related preferences of information seekers. Although they may increase the access to information, current AI tools reduce the diversity of information presentation [94], that our results show are a favorable characteristic of software documentation.

7.3 Future Research Directions

Our work focuses on the styling of documentation. However, much prior work has investigated the extent to which available *information* in software documentation is sufficient and useful to programmers [31, 167]. Additionally, prior research has involved the development of tools that allow programmers to augment existing information with their own annotations that can be shared with fellow information seekers [97, 235]. For example, a programmer may share annotations that highlight and clarify important, but complex parts of the resource. Future work can involve investigating the combination of information annotations and presentation customization to support programmers’ increased control to manipulate documentation resources to their usage context. We discuss ideas for future work related to *code example customization*, *querying pertinent documentation*, *other software documentation design considerations*, and the *communication between documentation creators and information seekers*.

7.3.1 Code Example Customization

Our research focuses primarily on software documentation resources as a whole. Although we report on the structure and organization properties of software tutorials (Chapter 4),

we do not investigate the characteristics of each of these elements in depth. Similarly, we compared the usage of different modalities and studied how programmers manipulated a multimodal tutorial to access particular modalities. However, each modality itself can be customized. For example, a for-loop in a code example can be written in a single line or across multiple lines. Additionally, the color scheme of code examples and font style of text content can be customized. The study of documentation properties to this level of detail is an avenue for future work.

In the domain of software engineering, code examples remain an integral learning tool, and searching specifically for code is an investigated research area [230]. Prior work has investigated what information about code examples are present in documents [45]. We gained insight on how useful regular versus summarized versus annotated code examples were for programmers completing different programming tasks. Future work can involve the enrichment of code examples present in the multimodal tutorial, such as by providing an in-resource executable environment, and customizable annotations.

Future work can also explore the variations in the extent of information that code examples can provide. As an example, in our design of multimodal documentation, we experimented with the *comprehensiveness* that a code example could provide. We developed two types of code examples, which we referred to as *focused* and *contiguous*. Focused code examples were short and associated with the topic of the related subsection of which they were a part. The intention of focused code examples was to allow a single topic section to be self-contained with multiple modalities that corresponded to one another. In contrast, *contiguous* code examples were long, because they amalgamated programming examples from all related subsections across the tutorial. The goal of contiguous code examples was to provide programmers who preferred to look only at code, a single location where all code related to the topic at hand was available. This could also help in comparing and contrasting syntaxes between multiple related programming concepts. However, this would result in a disconnection between different modalities, as all information would be present in each modality, sequentially. Thus, for our study of the multiple modalities (Chapter 6), we opted to remove this functionality and provided only a single way to view code examples in terms of their comprehensiveness. Future work can investigate whether allowing users to toggle between focused and contiguous code examples can support the search for pertinent code examples.

7.3.2 Querying Pertinent Documentation

The search for pertinent information involves the creation of text queries, that search engines can use to locate potentially relevant documentation resources. In our research, we do not focus on query formulation. One integral aspect of query formulation is the elicitation of both the `QUESTION` and the `PREFERENCE` that the programmer may have. However, this may not be trivial to elicit. For example, prior work has shown that programmers may even abandon the search for code examples, if they are unable to elicit their requirements clearly, and as a consequence, unable to find relevant examples [257]. Huang et al. recognized that there exists a lexical and knowledge gap between what a developer wants to achieve in

their programming task and the search query that will retrieve the exact information they need, despite semantically similar statements [100]. They created BIKER (Bi-Information source based Knowledge Recommendation) to recommend appropriate APIs based on the programming task context.

Prior work has explored the ability to retrieve information based on interaction behaviour of search results. Lu and Hsiao studied the searching behaviour of novice and advanced programmers and noted that novice programmers found difficulty in forming and refining search queries [149]. They developed the Personalized Information Seeking Assistant (PiSA) which recommended relevant terms to novice programmers to refine their search queries. White et al. developed an algorithm to identify and predict the top most relevant additional query terms that could be related to a searcher’s query based on their interaction with search results [262]. They conducted a study, comparing their “implicit feedback” approach wherein search results would be automatically reordered based on search behaviour, with a baseline wherein participants had to manually expand queries themselves, and manually click a button should they want to reorder results. They reported that there was at least a 67% overlap between the terms participants manually selected and their algorithm suggested, for four different search task types. Their participants described that the implicit feedback and automatic query updating would be more useful when their information need was not clear and may change, as opposed to when it was well-defined and constant. White et al. also concluded that the automated system relieved the user of too much control. Instead, providing recommendations through an automated system, but allowing users to make explicit choices on whether to incorporate those query recommendations in their search would retain control in the hands of the user. Macaw, developed by researchers at Microsoft, provides a way for users to ask questions and obtain answers [276]. Macaw allows the interaction through multiple modalities, including regular text, audio, PDF documents, and web page links. Such systems still rely on a user being extremely clear about what they want, need, and prefer and being able to formulate these specifications into a concise search query.

To support the search for information, current search engines rely largely on keyword-based approaches, such as the PageRank algorithm [184], which also leverage links between web pages and how the community accesses different web pages [132]. As a result, if a programmer searched for “inheritance in Java summarized code example”, such an algorithm would likely return results that matched the individual terms, rather than identify that “summarized code example” refers to the modality. Some search engines such as Google, allow users to *refine* their searches. For examples, users can specify the *type* of results, e.g. web pages, news articles, images, or videos, and the particular *sites* to search on [86]. Our findings about programmers’ preferences for information content and resource style indicate the need to make these preferences explicit when looking for information. Thus, future research on information seeking should involve the consideration of *contextual* information. Tools to support information seeking must be able to account for and identify preferences in documentation resource modalities, features, and other such design aspects from search queries.

7.3.3 Other Software Documentation Design Considerations

We studied the design of software documentation, with a specific focus on structure, organization, and presentation. However, a number of other aspects provide important context for documentation creation and use.

When seeking information, users may rely on their familiarity with documentors to identify pertinent resources, or may rely on how popular the source website is, as cues to search for information [138]. We noted from our third study phase (Chapter 5) that documentors are aware that users may rely on such cues, and thus make active efforts to become familiar with their audiences. For example, documentors may generate identifying content, e.g. a regular intro and/or outro in their Youtube channel, or pursue *growth and visibility* to gain popularity. Future work can investigate how aspects such as familiarity and popularity of documentation resources impact their use. Such research could provide insight on how the connection between documentors and programmers could be better supported, in addition to direct communication (see Section 7.3.4).

Another aspect that we noted was that the level of expertise of documentors and programmers impacted the creation and use of documentation, respectively. Lieberman argued that documentation creators should have an understanding of the knowledge documentation users have, to be able to tailor documentation appropriately [144]. Some documentors in our third study indicated that they were only able to describe information that was known to them, and thus selected topics that they were knowledgeable about. As a result, their documentation could cater to programmers who had less knowledge about the programming topic than the documentor. In our study of multimodal documentation (Chapter 6), we focused on creating prototype tutorials on basic concepts in Java. Future work can involve investigating how different modalities can be useful for learners of different levels of knowledge. Results from such an investigation would provide the necessary insights to determine whether multimodal, customizable documentation can be a feasible one-size-fits-all solution for different learning contexts.

The investigation of other layers of documentation customization are avenues for future work. For example, varying the level of detail across documents has been suggested to support beginners as they access technical documents [275]. In our construction of the prototype multimodal tutorials, we explored the ability to provide a configuration for showing “overview” versus “detailed” content, and thus varying the amount of information present through text. Future work can involve the evaluation of the usefulness of such additional customizable features that we had envisioned but could not evaluate, including providing global controls to show and hide all elements of a particular modality.

The findings from our research show promise for multimodal, customizable documentation that can cater to varying programmer needs as they search for information. Future work can investigate the amount of effort and resources it would require documentors to design and create such versatile documentation. Thus, an important avenue for future work is to gain insight on the extent to which the vision of multimodal, customizable documentation is feasible and appealing for both stakeholders, i.e. documentors and information seekers.

7.3.4 Communication Between Documentation Creators and Information Seekers

Prior work has show that the communication between developers and end users is integral to the success of a project, and that a breakdown of such communication can harm software projects [78]. Similarly, communication between documentation creators and information seekers can support information seekers in identifying pertinent resources to help them learn about software technologies.

Across the four phases of our research, we focused on the design of software documentation that acts as the form of interaction or *communication* [201] between documentation creators and information seekers. However, there is a need for a direct communication between both parties to support effective search and consumption of documentation resources. From our diary study in phase one, we noted that information seekers were forced to browse through multiple resources, to ultimately arrive at one that could fulfil their needs (see Section 3.5). During our interview study in phase three, documentors described the scalability issue with interacting with information seekers (see Section 5.5.2). As a consequence, documentation creators must use other proxies to identify what information seekers need, and information seekers must rely on implicit cues to manually identify resources that best suit their contexts. Our systematic frameworks from the first three phases of our research provide a foundation to facilitate a communication wherein information seekers can explicitly describe their needs and preferences, and documentation creators can explicitly describe their thought processes.

To *ask for* information about software technologies, currently, information seekers are able to connect to information providers primarily through asynchronous conversations such as on Stack Overflow or issue discussion threads. Such platforms host valuable information about software technologies [14, 27], but do not encourage immediate dialogue. Prior work has shown that live chat can be useful to discuss topics related to API usage [226]. However, they can also be a source of irrelevant comments that require further processing to identify only pertinent information [11]. Documentors in our third study indicated that they did not have the resources to manage an active communication with information seekers while prioritizing documentation creation (see Section 3.5), despite the enjoyment and learning that interacting with their audiences brings [252]. Further work can look into how to support the direct communication while overcoming challenges such as of coordination and scheduling, as well as mitigating spam and noise. Then, there may be potential for incorporating such communication as an optional, interactive modality in software documentation.

To *share* relevant context, prior work has proposed techniques to communicate developers' decisions to end users to provide context for software changes [4, 238]. Similarly for documentation, future work can study how to present the creation context in terms of mindsets and the corresponding considerations. This can support information seekers with related needs and preferences to being directed to documentation creators with those mindsets. Furthermore, information seekers can then directly request for particular information content or resource styles, or provide feedback to creators to consider. How to support such knowledge exchanges through a scalable, direct communication between documentation creators and information seekers, is an area that can be investigated in future work.

Chapter 8

Conclusion

Programmers have access to a large number of software documentation resources that provide information about software technologies, via different online platforms and media. Such documentation plays an important role in the development and maintenance of a software technology [277], without which the inner working of the technology is difficult to comprehend, leaving the user dealing with a black-box system [210]. Hence it is integral that programmers are able to retrieve the information pertinent to their needs to *understand* the software they develop, use, or maintain.

Navigating the set of available resources to find pertinent information can be time consuming, effort-intensive, and frustrating. Prior work has focused on identifying programmers' needs and creating efficient techniques to automatically retrieve information related to their needs. Yet, the use of documentation relies on the *context* in which a programmer searches for information, which can vary among programmers. Thus, documentation must be able to cater to these varying needs. Additionally, documentation creators, or documentors, have many decisions to make when designing software documentation, because of which creating and contributing documentation is a time- and effort-intensive process. Although much prior work has studied the value of software documentation, and how to automatically generate documentation resources, these studies focus on information content, neglecting the aspects of structure, organization, and presentation of software documentation. Additionally, there is little work that investigates how aspects of the documentation creation process and information seeking process correspond.

We studied the synergy between documentation and information seeking with the goal of supporting the design of a versatile documentation resource that can cater to varying programmer needs, while considering documentors' efforts. We approached software documentation through three lenses: the creation of documentation, the search for pertinent documentation, and the characteristics of documentation resources. With insights from empirical studies regarding these three perspectives, we proposed and evaluated a versatile design for software documentation. Our work proceeded in four phases.

First, we conducted a diary and interview study of ten programmers to understand how programmers find software learning resources online. We proposed a resource seeking model that captures systematically how programmers make decisions when navigating between on-

line learning resources for a software technology. Specifically, we found that programmers can have PREFERENCES of the kind of RESOURCES they are looking for, which may be backed by BELIEFS from previous experiences. These two components provide context for the QUESTIONS that programmers are looking to solve. Furthermore, programmers use various CUES to select RESOURCES to click on during their search, and evaluate them based on certain IMPRESSION FACTORS. We elicited nine relations between the six components of our model, and investigated the nature of these relations. We found, for example, that participants depended on the CUE *Familiarity* to select RESOURCES that are *Forums* more than expected.

Next, we examined the extent to which properties of online technology resources vary in the five programming languages: Java, C#, Python, Javascript, and Typescript. Our observations of property distributions and correlations revealed that resources cannot be characterized by their properties in isolation, and in a mutually exclusive manner. We propose the representation of resources by their properties that deviate from the norm as *distinguishing attributes*. We formalized the concept of a *resource style* as a combination of co-occurring distinguishing attributes, as part of a framework to characterize resources based on their design. We leveraged our framework to implement three techniques to identify relevant resource styles and applied these techniques on our data set of 2551 resources. We discovered that no resource style in any particular programming language is more notable than the rest. The variety of styles observed indicate that there is a wide range of design choices for resource creators and seekers.

Third, we interviewed 26 documentors who voluntarily created text or video tutorials about software technologies. We asked them about *why* and *how* they contributed documentation. We elicited documentors' *considerations* during the documentation contribution process along three *dimensions*: five *motivations*, five *topic selection techniques*, and six *styling objectives*. We grouped related considerations based on their common implicit themes to elicit five *software documentor mindsets*. For example, some documentors may focus on the *novelty and value addition* of the documentation they contribute, inspired from their prior experiences with inadequate documentation. Our findings suggest the need to support documentors in balancing multiple mindsets: incorporating their own needs, preferences, and learning, contributing original content, reaching the audience, while considering non-human logistics such as suitability of the format to a topic.

Finally, given our prior findings that programmers 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 across topics. 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 documentation resource can serve diverse information needs for programming tasks.

Our contributions include a model for resource seeking behaviour and a model to capture thought processes during documentation creation. Furthermore, we contribute a framework for characterizing documentation resources based on their varying properties, as well as a prototype tutorial to capture documentation variations as modalities for programmers to choose from. Our studies augment existing literature on assisting programmers seeking information about a technology. Our findings are also fundamental to resource creators in understanding their target population's search behaviour and how the resources they create can satisfy programmers' preferences and needs. Additionally, our observations can also inform search tools to alleviate time and effort spent in search.

The work in this thesis lies on the intersection between *document request* and *document generation*, both directions that require advancement to build better quality, user-catered documentation according to Robillard et al. [211]. Our research is an initial step towards the user interface of multimodal, multifeatured, customizable software documentation. Consequently, documentors will be able to focus on curating the information that corresponds to their mindsets, and information seekers can systematically identify documentation from documentors who cater to their specific resource needs and preferences. We plan to continue to work towards the design and development of versatile software documentation that retains the creative and experienced human-in-the-loop, to support effective learning of software technologies.

Resource References

1. https://learnpython.org/en/Numpy_Arrays
2. https://learnpython.org/en/Pandas_Basics
3. <https://www.javatpoint.com/difference-between-namespaces-and-modules>
4. <https://www.javatpoint.com/convert-object-to-array-in-javascript>
5. <https://docs.python.org/3/tutorial/appendix.html>
6. <https://docs.oracle.com/javase/tutorial/essential/regex/quant.html>
7. <https://docs.oracle.com/javase/tutorial/collections/interfaces/order.html>
8. https://www.tutorialspoint.com/javascript/javascript_operators.htm
9. <https://www.w3schools.blog/union-type-typescript>
10. <https://www.javatpoint.com/instance-initializer-block>
11. <https://www.javatpoint.com/how-to-enable-javascript-in-my-browser>
12. <https://www.javatpoint.com/design-patterns-c-sharp>
13. <https://www.guru99.com/date-time-and-datetime-classes-in-python.html>
14. https://www.tutorialspoint.com/javascript/javascript_events.htm
15. <http://csharp.net-informations.com/gui/cs-scrollbars.htm>
16. <https://www.educba.com/logical-operators-in-c-sharp/>
17. <https://docs.oracle.com/javase/tutorial/collections/interfaces/order.html>
18. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures
19. <https://www.tutorialkart.com/typescript/typescript-switch->
20. https://www.tutorialspoint.com/python/python_sending_email.htm
21. <https://www.guru99.com/java-platform.html>
22. <https://www.javatpoint.com/how-to-install-python>
23. <http://csharp.net-informations.com/statements/enum.htm>
24. https://www.tutorialspoint.com/java/java_quick_guide.htm

25. <https://www.w3schools.blog/class-in-typescript>
26. <https://www.typescripttutorial.net/typescript-tutorial/typescript-class->
27. <https://www.geeksforgeeks.org/switch-statement-in-c-sharp->
28. <https://www.tutorialsteacher.com/csharp/csharp-switch>

Bibliography

- [1] 2009. IEEE Standard for Information Technology–Systems Design–Software Design Descriptions. *IEEE STD 1016-2009* (2009).
- [2] 2024. A two-actor model for understanding user engagement with content creators: Applying social capital theory. *Computers in Human Behavior* 156 (2024). <https://doi.org/10.1016/j.chb.2024.108237>
- [3] Hervé Abdi et al. 2007. Bonferroni and Šidák Corrections for Multiple Comparisons. *Encyclopedia of Measurement and Statistics* 3 (2007), 103–107.
- [4] Ulrike Abelein and Barbara Paech. 2012. A proposal for enhancing user-developer communication in large IT projects. In *Proceedings of the International Workshop on Co-operative and Human Aspects of Software Engineering*. 1–3. <https://doi.org/10.1109/CHASE.2012.6223014>
- [5] Iyadunni J. Adenuga, Benjamin V. Hanrahan, Chen Wu, and Prasenjit Mitra. 2022. Living Documents: Designing for User Agency over Automated Text Summarization. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*. 1–6. <https://doi.org/10.1145/3491101.3519810>
- [6] Emad Aghajani, Csaba Nagy, Mario Linares-Vásquez, Laura Moreno, Gabriele Bavota, Michele Lanza, and David C. Shepherd. 2020. Software Documentation: The Practitioners’ Perspective. In *Proceedings of the International Conference on Software Engineering*. <https://doi.org/10.1145/3377811.3380405>
- [7] Emad Aghajani, Csaba Nagy, Olga Lucero Vega-Márquez, Mario Linares-Vásquez, Laura Moreno, Gabriele Bavota, and Michele Lanza. 2019. Software Documentation Issues Unveiled. In *Proceedings of International Conference on Software Engineering*. <https://doi.org/10.1109/ICSE.2019.00122>
- [8] George Ajam, Carlos Rodriguez, and Boualem Benatallah. 2021. Scout-bot: Leveraging API Community Knowledge for Exploration and Discovery of API Learning Resources. *CLEI Electronic Journal* 24, 2 (2021). <https://doi.org/10.19153/cleiej.24.2.5>
- [9] Ra’Fat Al-Msie’Deen, Abdelhak Seriai, Marianne Huchard, Christelle Urtado, Sylvain Vauttier, and Hamzeh Eyal-Salman. 2013. Mining Features from the Object-Oriented

- Source Code of a Collection of Software Variants Using Formal Concept Analysis and Latent Semantic Indexing. In *Proceedings of the International Conference on Software Engineering and Knowledge Engineering*.
- [10] Ra'Fat AL-Msie'deen, Abdelhak Seriai, Marianne Huchard, Christelle Urtado, Sylvain Vauttier, and Hamzeh Eyal Salman. 2013. Feature Location in a Collection of Software Product Variants Using Formal Concept Analysis. In *Safe and Secure Software Reuse*. https://doi.org/10.1007/978-3-642-38977-1_22
 - [11] Mohammad D Alahmadi, Khalid T Mursi, Mohammed A Alqarni, Ahmad J Tayeb, and Faisal S Alsubaei. 2024. Analyzing and categorization developer intent on Twitch live chat. *Programming and Computer Software* 50, 5 (2024), 392–402. <https://doi.org/10.1134/S0361768824700191>
 - [12] Gianni Angelini. 2018. Current Practices in Web API Documentation. In *European Academic Colloquium on Technical Communication*.
 - [13] Renana Arizon-Peretz, Irit Hadar, Gil Luria, and Sofia Sherman. 2021. Understanding Developers' Privacy and Security Mindsets via Climate Theory. *Empirical Software Engineering* 26, 123 (2021). <https://doi.org/10.1007/s10664-021-09995-z>
 - [14] Deeksha Arya, Wenting Wang, Jin LC Guo, and Jinghui Cheng. 2019. Analysis and detection of information types of open source software issue discussions. In *Proceedings of the International Conference on Software Engineering*. 454–464.
 - [15] Deeksha M. Arya, Jin L. C. Guo, and Martin P. Robillard. 2020. Information Correspondence Between Types of Documentation for APIs. *Empirical Software Engineering* 25 (2020), 4069–4096. <https://doi.org/10.1007/s10664-020-09857-0>
 - [16] Deeksha M. Arya, Jin L. C. Guo, and Martin P. Robillard. 2022. How Programmers Find Online Learning Resources. *Empirical Software Engineering* 28, 3 (2022). <https://doi.org/10.1007/s10664-022-10246-y>
 - [17] Deeksha M. Arya, Jin L. C. Guo, and Martin P. Robillard. 2022. Replication package for *How Programmers Find Online Learning Resources*. <https://doi.org/10.5281/zenodo.7504510>
 - [18] Deeksha M. Arya, Jin L. C. Guo, and Martin P. Robillard. 2023. Replication package for *Properties and Styles of Software Technology Tutorials*. <https://doi.org/10.5281/zenodo.10048532>
 - [19] Deeksha M. Arya, Jin L. C. Guo, and Martin P. Robillard. 2024. Properties and Styles of Software Technology Tutorials. *IEEE Transactions on Software Engineering* 50, 2 (2024). <https://doi.org/10.1109/TSE.2023.3332568>
 - [20] Deeksha M. Arya, Jin L. C. Guo, and Martin P. Robillard. 2024. Replication pack-

- age for *The Software Documentor Mindset*. <https://doi.org/10.5281/zenodo.14368203>
- [21] Deeksha M. Arya, Jin L. C. Guo, and Martin P. Robillard. 2024. The Software Documentor Mindset. <https://arxiv.org/abs/2412.09422>
- [22] Deeksha M. Arya, Jin L. C. Guo, and Martin P. Robillard. 2024. Why People Contribute Software Documentation. In *Proceedings of the International Conference on Cooperative and Human Aspects of Software Engineering*. <https://doi.org/10.1145/3641822.3641881>
- [23] Deeksha M. Arya, Jin L. C. Guo, and Martin P. Robillard. 2025. How Programmers Interact with Multimodal Software Documentation. In *Proceedings of the International Conference on Cooperative and Human Aspects of Software Engineering*. <https://doi.org/10.1109/CHASE66643.2025.00029>
- [24] Deeksha M. Arya, Mathieu Nassif, and Martin P. Robillard. 2022. A Data-centric Study of Software Tutorial Design. *IEEE Software* 39, 3 (2022). <https://doi.org/10.1109/MS.2021.3090978>
- [25] Shams Azad, Peter C. Rigby, and Latifa Guerrouj. 2017. Generating Api Call Rules from Version History and Stack Overflow Posts. *Transactions on Software Engineering and Methodology* 25, 4 (2017). <https://doi.org/10.1145/2990497>
- [26] Gina R. Bai, Joshua Kayani, and Kathryn T. Stolee. 2020. How Graduate Computing Students Search When Using an Unfamiliar Programming Language. In *Proceedings of the International Conference on Program Comprehension*. 160–171. <https://doi.org/10.1145/3387904.3389274>
- [27] Ohad Barzilay, Christoph Treude, and Alexey Zagalsky. 2013. Chapter 15: Facilitating crowd sourced software engineering via Stack Overflow. In *Finding source code on the web for remix and reuse*. Springer. <https://doi.org/10.1007/978-1-4614-6596-6>
- [28] Marcia J. Bates. 1990. Where Should The Person Stop And The Information Search Interface Start? *Information Processing and Management* 26, 5 (1990), 575–591.
- [29] Katharina Bernecker and Veronika Job. 2019. Mindset Theory. In *Social Psychology in Action: Evidence-Based Interventions from Theory to Practice*. Springer, 179–191. https://doi.org/10.1007/978-3-030-13788-5_12
- [30] Avinash Bhat, Disha Shrivastava, and Jin L.C. Guo. 2024. Do LLMs Meet the Needs of Software Tutorial Writers? Opportunities and Design Implications. In *Proceedings of the ACM Conference on Designing Interactive Systems*. <https://doi.org/10.1145/3643834.3660692>
- [31] Jared Bhatti, Sarah Corleissen, Jen Lambourne, David Nunez, and Heidi Waterhouse.

2021. Measuring Documentation Quality. In *Docs for Developers: An Engineer's Field Guide to Technical Writing*. Apress, 179–191. https://doi.org/10.1007/978-1-4842-7217-6_9
- [32] Jared Bhatti, Sarah Corleissen, Jen Lambourne, David Nunez, and Heidi Waterhouse. 2021. Understanding Your Audience. In *Docs for Developers: An Engineer's Field Guide to Technical Writing*. Apress, 1–21. https://doi.org/10.1007/978-1-4842-7217-6_1
- [33] Christina Bottomley. 2005. What Part Writer? What Part Programmer? A Survey of Practices And Knowledge Used in Programmer Writing. In *Proceedings of the International Professional Communication Conference*. <https://doi.org/10.1109/IPCC.2005.1494255>
- [34] 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 Conference on Human Factors in Computing Systems*. 1589–1598. <https://doi.org/10.1145/1518701.1518944>
- [35] Suzanne Briet. 1951. *Qu'est-ce que la documentation?* Éditions Documentaires, Industrielles et Techniques.
- [36] Ash Buchanan. 2024. What is Mindset? 100 Definitions from the Field. In *Handbook of Mindset Research*. <https://doi.org/10.31234/osf.io/5xeqv>
- [37] Frank Buchli. 2003. *Detecting Software Patterns using Formal Concept Analysis*. Master's thesis.
- [38] Sven Buschbeck, Anthony Jameson, Adrian Spirescu, Tanja Schneeberger, Raphaël Troncy, Houda Khrouf, Osma Suominen, and Eero Hyvönen. 2013. Parallel Faceted Browsing. In *Proceedings of the Extended Abstracts on Human Factors in Computing Systems*. 3023–3026. <https://doi.org/10.1145/2468356.2479601>
- [39] Raymond P.L. Buse and Westley R. Weimer. 2010. Learning a Metric for Code Readability. *IEEE Transactions on Software Engineering* (2010). <https://doi.org/10.1109/TSE.2009.70>
- [40] Greg Butler, Peter Grogono, and Ferhat Khendek. 1998. A Reuse Case Perspective On Documenting Frameworks. In *Proceedings of the Asia Pacific Software Engineering Conference*. 94–101. <https://doi.org/10.1109/APSEC.1998.733596>
- [41] Greg Butler, Rudolf K Keller, and Hafedh Mili. 2000. A Framework For Framework Documentation. *Computing Surveys (ACM)* (2000). <https://doi.org/10.1145/351936.351951>
- [42] Alessandro Caponi, Angelo Di Iorio, Fabio Vitali, Paolo Alberti, and Marcello Scatá.

2018. Exploiting Patterns and Templates for Technical Documentation. In *Proceedings of the ACM Symposium on Document Engineering*. Article 30. <https://doi.org/10.1145/3209280.3209537>
- [43] John M. Carroll. 1990. An Overview of Minimalist Instruction. In *Proceedings of the Annual Hawaii International Conference on System Sciences*, Vol. 4. 210–219. <https://doi.org/10.1109/HICSS.1990.205259>
- [44] A. Castellanos, J. Cigarrán, and A. García-Serrano. 2017. Formal Concept Analysis for Topic Detection: A Clustering Quality Experimental Analysis. *Information Systems* 66 (2017), 24–42. <https://doi.org/10.1016/j.is.2017.01.008>
- [45] Preetha Chatterjee, Manziba Akanda Nishi, Kostadin Damevski, Vinay Augustine, Lori Pollock, and Nicholas A. Kraft. 2017. What Information About Code Snippets is Available in Different Software Related Documents? An Exploratory Study. In *Proceedings of the International Conference on Software Analysis, Evolution and Reengineering*. <https://doi.org/10.1109/SANER.2017.7884638>
- [46] Souti Chattopadhyay, Nicholas Nelson, Audrey Au, Natalia Morales, Christopher Sanchez, Rahul Pandita, and Anita Sarma. 2020. A Tale from the Trenches: Cognitive Biases and Software Development. In *Proceedings of the International Conference on Software Engineering*. 654–665. <https://doi.org/10.1145/3377811.3380330>
- [47] Souti Chattopadhyay, Nicholas Nelson, Thien Nam, McKenzie Calvert, and Anita Sarma. 2018. Context in Programming: An Investigation of How Programmers Create Context. In *Proceedings of the International Workshop on Cooperative and Human Aspects of Software Engineering*. 33–36. <https://doi.org/10.1145/3195836.3195861>
- [48] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebggen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating Large Language Models Trained on Code. *arXiv:2107.03374* (2021).
- [49] Shi-Yi Chen, Zhe Feng, and Xiaolian Yi. 2017. A General Introduction to Adjustment for Multiple Comparisons. *Journal of Thoracic Disease* 9, 6 (2017). <https://doi.org/10.21037/jtd.2017.05.34>

- [50] Juan Cigarran, Angel Castellanos, and Ana Garcia-Serrano. 2016. A Step Forward for Topic Detection in Twitter: An FCA-based Approach. *Expert Systems with Applications* 57 (2016), 21–36. <https://doi.org/10.1016/j.eswa.2016.03.011>
- [51] Juan M. Cigarrán, Julio Gonzalo, Anselmo Peñas, and Felisa Verdejo. 2004. Browsing Search Results via Formal Concept Analysis: Automatic Selection of Attributes. In *Concept Lattices*, Vol. 2961. 74–87. https://doi.org/10.1007/978-3-540-24651-0_8
- [52] Claudio G. Cortese. 2005. Learning Through Teaching. *Management Learning* 36, 1 (2005). <https://doi.org/10.1177/1350507605049905>
- [53] John W. Creswell. 2014. *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*. SAGE Publications.
- [54] Barthélémy Dagenais and Martin P. Robillard. 2010. Creating and Evolving Developer Documentation: Understanding the Decisions of Open Source Contributors. In *Proceedings of Foundations of Software Engineering*. <https://doi.org/10.1145/1882291.1882312>
- [55] Barthélémy Dagenais and Martin P. Robillard. 2014. Using Traceability Links to Recommend Adaptive Changes for Documentation Evolution. *IEEE Transactions on Software Engineering* 40, 11 (2014), 1126–1146. <https://doi.org/10.1109/TSE.2014.2347969>
- [56] Sergio Cozzetti B. de Souza, Nicolas Anquetil, and Káthia M. de Oliveira. 2005. A Study of the Documentation Essential to Software Maintenance. In *Proceedings of the 23rd Annual International Conference on Design of Communication: Documenting & Designing for Pervasive Information*. 68–75. <https://doi.org/10.1145/1085313.1085331>
- [57] Steve Delanghe. 2000. Using Learning Styles in Software Documentation. *Transactions on Professional Communication* 43, 2 (2000). <https://doi.org/10.1109/47.843647>
- [58] Sérgio M. Dias and Newton J. Vieira. 2015. Concept Lattices Reduction: Definition, Analysis and Classification. *Expert Systems with Applications* 42 (2015), 7084–7097. Issue 20. <https://doi.org/10.1016/j.eswa.2015.04.044>
- [59] Pierpaolo Dondio and Suha Shaheen. 2019. Is Stack Overflow an Effective Complement to Gaining Practical Knowledge Compared to Traditional Computer Science Learning?. In *Proceedings of the International Conference on Education Technology and Computers*. 132–138. <https://doi.org/10.1145/3369255.3369258>
- [60] Ekwa Duala-Ekoko and Martin P Robillard. 2012. Asking and Answering Questions about Unfamiliar APIs: An Exploratory Study. In *Proceedings of the International Conference on Software Engineering*. 266–276. <https://doi.org/10.1109/ICSE.2012.6227187>

- [61] Koznov D.V., Luciv D.V., and Chernishev G.A. 2017. Duplicate Management in Software Documentation Maintenance. In *Proceedings of the International Conference on Actual Problems of System and Software Engineering*.
- [62] Ralph H Earle, Mark A Rosso, and Kathryn E Alexander. 2015. User Preferences of Software Documentation Genres. In *Proceedings of the Annual International Conference on the Design of Communication*. Article 46. <https://doi.org/10.1145/2775441.2775457>
- [63] Lisa Ede and Andrea Lunsford. 1984. Audience Addressed / Audience Invoked: The Role of Audience in Composition Theory And Pedagogy. *College Composition and Communication* 35, 2 (1984).
- [64] Mathias Ellmann. 2017. On the similarity of software development documentation. In *Proceedings of the 11th Joint Meeting on Foundations of Software Engineering*. 1030–1033. <https://doi.org/10.1145/3106237.3119875>
- [65] Jennifer English, Marti Hearst, Rashmi Sinha, Kirsten Swearingen, and Ka-Ping Yee. 2002. Hierarchical Faceted Metadata in Site Search Interfaces. In *Proceedings of the Extended Abstracts on Human Factors in Computing Systems*. 628–639. <https://doi.org/10.1145/506443.506517>
- [66] Ali Erdem, W Lewis Johnson, and Stacy Marsella. 1998. Task Oriented Software Understanding. In *Proceedings of the International Conference on Automated Software Engineering*. 230–239. <https://doi.org/10.1109/ASE.1998.732658>
- [67] Katalin Erdos and Harry M. Sneed. 1998. Partial Comprehension of Complex Programs (Enough to Perform Maintenance). In *Proceedings of the International Workshop on Program Comprehension*. 98–105. <https://doi.org/10.1109/WPC.1998.693322>
- [68] Neil Ernst, Alexander Borgida, Ivan J. Jureta, and John Mylopoulos. 2014. An Overview of Requirements Evolution. In *Evolving Software Systems*. 3–32. https://doi.org/10.1007/978-3-642-45398-4_1
- [69] Javier Escobar-Avila, Deborah Venuti, Massimiliano Di Penta, and Sonia Haiduc. 2019. 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*. 170–181. <https://doi.org/10.1109/ICSE-SEET.2019.00026>
- [70] Angela Fan, Beliz Gokkaya, Mark Harman, Mitya Lyubarskiy, Shubho Sengupta, Shin Yoo, and Jie M. Zhang. 2023. Large Language Models for Software Engineering: Survey and Open Problems. In *Proceedings of the International Conference on Software Engineering: Future of Software Engineering*. 31–53. <https://doi.org/10.1109/ICSE-FoSE59343.2023.00008>

- [71] Sébastien Ferré, Marianne Huchard, Mehdi Kaytoue, Sergei O. Kuznetsov, and Amedeo Napoli. 2020. Formal Concept Analysis: From Knowledge Discovery to Knowledge Processing. In *A Guided Tour of Artificial Intelligence Research: Volume II: AI Algorithms*. https://doi.org/10.1007/978-3-030-06167-8_13
- [72] Carmen Fischer, Charlotte P Malycha, and Ernestine Schafmann. 2019. The Influence of Intrinsic Motivation and Synergistic Extrinsic Motivators on Creativity and Innovation. *Frontiers in psychology* 10 (2019). <https://doi.org/10.3389/fpsyg.2019.00137>
- [73] Andrew Forward and Timothy C Lethbridge. 2002. The relevance of software documentation, tools and technologies. In *Proceedings of the ACM Symposium on Document Engineering*. <https://doi.org/10.1145/585058.585065>
- [74] Adam Fourney and Michael Terry. 2014. Mining Online Software Tutorials: Challenges and Open Problems. In *Proceedings of the Conference on Human Factors in Computing Systems*. <https://doi.org/10.1145/2559206.2578862>
- [75] Jessie Galasso-Carbonnel, Marianne Huchard, André Miralles, and Clémentine Nebut. 2017. Feature Model Composition Assisted by Formal Concept Analysis. In *Proceedings of the International Conference on Evaluation of Novel Approaches to Software Engineering*. <https://doi.org/10.5220/0006276600270037>
- [76] Jessie Galasso-Carbonnel, Marianne Huchard, and Clémentine Nebut. 2017. Analyzing Variability in Product Families through Canonical Feature Diagrams. In *Proceedings of the International Conference on Software Engineering and Knowledge Engineering*. <https://doi.org/10.18293/SEKE2017-087>
- [77] Rosalva E Gallardo-Valencia and Susan Elliott Sim. 2011. What Kinds of Development Problems can be Solved by Searching the Web?: A Field Study. In *Proceedings of the International Conference on Software Engineering*. 41–44. <https://doi.org/10.1145/1985429.1985440>
- [78] Michael J Gallivan and Mark Keil. 2003. The user–developer communication process: A critical case study. *Information Systems Journal* 13, 1 (2003), 37–68. <https://doi.org/10.1046/j.1365-2575.2003.00138.x>
- [79] Bernhard Ganter, Rudolf Wille, and C. Franzke. 1997. *Formal Concept Analysis: Mathematical Foundations*. Springer-Verlag. <https://doi.org/10.1007/978-3-642-59830-2>
- [80] Marco Gerosa, Igor Wiese, Bianca Trinkenreich, Georg Link, Gregorio Robles, Christoph Treude, Igor Steinmacher, and Anita Sarma. 2021. The Shifting Sands of Motivation: Revisiting What Drives Contributors in Open Source. In *International Conference on Software Engineering*. 1046–1058. <https://doi.org/10.1109/ICSE43902.2021.00098>

-
- [81] Alastair Gill, Scott Nowson, and Jon Oberlander. 2009. What Are They Blogging About? Personality, Topic and Motivation in Blogs. In *Proceedings of the International AAAI Conference on Web and Social Media*. <https://doi.org/10.1609/icwsm.v3i1.13949>
- [82] Ellen R Girden. 1992. *ANOVA: Repeated measures*. Sage.
- [83] Robert Godin and Hamed Mili. 1993. Building and Maintaining Analysis-Level Class Hierarchies Using Galois Lattices. In *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications*. <https://doi.org/10.1145/165854.165931>
- [84] Peter M. Gollwitzer. 2012. Mindset Theory of Action Phases. In *Handbook of Theories of Social Psychology: Volume 1*. SAGE Publications Ltd, 526–546. <https://doi.org/10.4135/9781446249215.n26>
- [85] David P. Goodwin. 1991. Emplotting the Reader: Motivation and Technical Documentation. *Journal of Technical Writing and Communication* 21 (1991). Issue 2. <https://doi.org/10.2190/1TLD-2JBL-DD7X-PXK3>
- [86] Google. Accessed: November 25th, 2024. Refine Google Searches. <https://support.google.com/websearch/answer/2466433>
- [87] Philip J. Guo. 2017. Older Adults Learning Computer Programming: Motivations, Frustrations, and Design Opportunities. In *Proceedings of the Conference on Human Factors in Computing Systems*. <https://doi.org/10.1145/3025453.3025945>
- [88] Carl Gutwin, Andy Cockburn, and Nickolas Gough. 2017. A Field Experiment of Spatially-Stable Overviews for Document Navigation. In *Proceedings of the Conference on Human Factors in Computing Systems*. 5905–5916. <https://doi.org/10.1145/3025453.3025905>
- [89] Irit Hadar, Tomer Hasson, Oshrat Ayalon, Michael Birnhack, Sofia Sherman, and Arod Balissa. 2018. Privacy by Designers: Software Developers’ Privacy Mindset. *Empirical Software Engineering* 23 (2018). <https://doi.org/10.1007/s10664-017-9517-1>
- [90] Tom Hanika, Maren Koyda, and Gerd Stumme. 2018. Relevant Attributes in Formal Contexts. (2018). https://doi.org/10.1007/978-3-030-23182-8_8
- [91] Andrew Head, Jason Jiang, James Smith, Marti A Hearst, and Björn Hartmann. 2020. Composing Flexibly-Organized Step-by-Step Tutorials from Linked Source Code, Snippets, and Outputs. In *Proceedings of the Conference on Human Factors in Computing Systems*. 1–12. <https://doi.org/10.1145/3313831.3376798>
- [92] Ava Heinonen, Bettina Lehtelä, Arto Hellas, and Fabian Fagerholm. 2023. Synthesizing Research on Programmers’ Mental Models of Programs, Tasks and Concepts — A Systematic Literature Review. *Information and Software Technology* 164 (2023). <https://doi.org/10.1016/j.infsof.2023.107300>

- [93] E. Tory Higgins and Abigail A. Scholer. 2009. Engaging the Consumer: The Science and Art of the Value Creation Process. *Journal of Consumer Psychology* (2009). <https://doi.org/10.1016/j.jcps.2009.02.002>
- [94] Noora Hirvonen, Ville Jylhä, Yucong Lao, and Stefan Larsson. 2024. Artificial intelligence in the information ecosystem: Affordances for everyday information seeking. *Journal of the Association for Information Science and Technology* 75, 10 (2024), 1152–1165.
- [95] E. Hoque and G. Carenini. 2014. ConVis: A Visual Text Analytic System for Exploring Blog Conversations. *Computer Graphics Forum* 33, 3 (2014), 221–230. <https://doi.org/10.1111/cgf.12378>
- [96] Andre Hora. 2021. Googling for Software Development: What Developers Search For and What They Find. In *Proceedings of the International Conference on Mining Software Repositories*. 317–328. <https://doi.org/10.1109/MSR52588.2021.00044>
- [97] Amber Horvath, Michael Xieyang Liu, River Hendriksen, Connor Shannon, Emma Paterson, Kazi Jawad, Andrew Macvean, and Brad A Myers. 2022. Understanding How Programmers Can Use Annotations on Documentation. In *Proceedings of the Conference on Human Factors in Computing Systems*. Article 69. <https://doi.org/10.1145/3491102.3502095>
- [98] Xinyi Hou, Yanjie Zhao, Yue Liu, Zhou Yang, Kailong Wang, Li Li, Xiapu Luo, David Lo, John Grundy, and Haoyu Wang. 2024. Large Language Models for Software Engineering: A Systematic Literature Review. *Transactions on Software Engineering and Methodology* (2024). <https://doi.org/10.1145/3695988>
- [99] Siw Elisabeth Hove and Bente Anda. 2005. Experiences from Conducting Semi-structured Interviews in Empirical Software Engineering Research. In *International Software Metrics Symposium*. <https://doi.org/10.1109/METRICS.2005.24>
- [100] Qiao Huang, Xin Xia, Zhenchang Xing, David Lo, and Xinyu Wang. 2018. API Method Recommendation Without Worrying About the Task-API Knowledge Gap. In *Proceedings of the International Conference on Automated Software Engineering*. 293–304. <https://doi.org/10.1145/3238147.3238191>
- [101] Sheng-Bo Huang, Yu-Lin Jeng, and Chin-Feng Lai. 2021. Note-taking Learning System: The Use of the Learning Style Theory and the Peer Learning Method on Computer Programming Course. *Journal of Educational Computing Research* 59 (2021). Issue 5. <https://doi.org/10.1177/0735633120985235>
- [102] William Hudson. 2013. Chapter 22 - Card sorting. In *The Encyclopedia of Human-Computer Interaction*. Interaction Design Foundation.
- [103] Shin-Yuan Hung, Alexandra Durcikova, Hui-Min Lai, and Wan-Mei Lin. 2011. The

- Influence of Intrinsic and Extrinsic Motivation on Individuals Knowledge Sharing Behavior. *International Journal of Human-Computer Studies* 69 (2011). Issue 6. <https://doi.org/j.ijhcs.2011.02.004>
- [104] Matthew Hurst. 2006. Towards a Theory of Tables. *International Journal of Document Analysis and Recognition* 8 (2006), 123–131. <https://doi.org/10.1007/s10032-006-0016-y>
- [105] Dmitry I. Ignatov. 2015. Introduction to Formal Concept Analysis and its Applications in Information Retrieval and Related Fields. Communications in Computer and Information Science, Vol. 505. 42–141. https://doi.org/10.1007/978-3-319-25485-2_3
- [106] Sergio Inzunza, Reyes Juárez-Ramírez, and Samantha Jiménez. 2018. API documentation: A conceptual evaluation model. In *Advances in Intelligent Systems and Computing*. https://doi.org/10.1007/978-3-319-77712-2_22
- [107] Leonardo Horn Iwaya, Muhammad Ali Babar, and Awais Rashid. 2023. Privacy Engineering in the Wild: Understanding the Practitioners’ Mindset, Organizational Aspects, and Current Practices. *Transactions on Software Engineering* 49, 9 (2023). <https://doi.org/10.1109/TSE.2023.3290237>
- [108] Riitta Jääskeläinen. 2010. Think-aloud Protocol. *Handbook of translation studies* 1 (2010), 371–374.
- [109] Mohieddin Jafari and Naser Ansari-Pour. 2018. Why, When and How to Adjust Your P Values? *Cell Journal (Yakhteh)* 20, 4 (2018). <https://doi.org/10.22074/cellj.2019.5992>
- [110] Nicolas Jay, François Kohler, and Amedeo Napoli. 2008. Analysis of Social Communities with Iceberg and Stability-Based Concept Lattices. In *Formal Concept Analysis (Lecture Notes in Computer Science)*. https://doi.org/10.1007/978-3-540-78137-0_19
- [111] He Jiang, Jingxuan Zhang, Zhilei Ren, and Tao Zhang. 2017. An Unsupervised Approach for Discovering Relevant Tutorial Fragments for APIs. In *Proceedings of the International Conference on Software Engineering*. 38–48. <https://doi.org/10.1109/ICSE.2017.12>
- [112] Xiaoyu Jin, Nan Niu, and Michael Wagner. 2017. 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*. 31–35. <https://doi.org/10.1109/VLHCC.2017.8103447>
- [113] Ralph E Johnson. 1992. Documenting Frameworks Using Patterns. In *Proceedings of the Conference on Object-oriented Programming Systems, Languages, and Applications*. <https://doi.org/10.1145/141936.141943>

- [114] Jitendra Josyula, Sarat Panamgipalli, Muhammad Usman, Ricardo Britto, and Naurman Bin Ali. 2018. Software Practitioners’ Information Needs and Sources: A Survey Study. In *Proceedings of the International Workshop on Empirical Software Engineering in Practice*. 1–6. <https://doi.org/10.1109/IWESEP.2018.00009>
- [115] Yongnam Jung, Cheng Chen, Eunhae Jang, and S. Shyam Sundar. 2024. Do We Trust ChatGPT as much as Google Search and Wikipedia?. In *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*. Article 111, 9 pages. <https://doi.org/10.1145/3613905.3650862>
- [116] Mika Käki. 2005. Findex: Search Result Categories Help Users When Document Ranking Fails. In *Proceedings of the Conference on Human Factors in Computing Systems*. 131–140. <https://doi.org/10.1145/1054972.1054991>
- [117] Mika Käki and Anne Aula. 2005. Findex: Improving Search Result Use Through Automatic Filtering Categories. *Interacting with Computers* 17 (2005), 187–206. Issue 2. <https://doi.org/10.1016/j.intcom.2005.01.001>
- [118] Joyce Karreman and Nicole Loorbach. 2013. Use and Effect of Motivational Elements in User Instructions: What We Do and Don’t Know. In *Proceedings of the International Professional Communication Conference*. <https://doi.org/10.1109/IPCC.2013.6623940>
- [119] J. Karreman, N. Ummelen, and M. Steehouder. 2005. Procedural and Declarative Information in User Instructions: What We Do and Don’t Know About These Information Types. In *Proceedings of the International Professional Communication Conference*. <https://doi.org/10.1109/IPCC.2005.1494193>
- [120] Ranim Khojah, Mazen Mohamad, Philipp Leitner, and Francisco Gomes de Oliveira Neto. 2024. Beyond Code Generation: An Observational Study of ChatGPT Usage in Software Engineering Practice. *Proceedings of the ACM on Software Engineering* 1, Article 81 (2024), 22 pages. <https://doi.org/10.1145/3660788>
- [121] Jong W Kim, Frank E Ritter, and Richard J Koubek. 2011. An Integrated Theory for Improved Skill Acquisition and Retention in the Three Stages of Learning. *Theoretical Issues in Ergonomics Science* 14, 1 (2011). <https://doi.org/10.1080/1464536X.2011.573008>
- [122] J. Peter Kincaid, Robert P. Fishburne, Richard L. Rogers, and Brad S. Chissom. 1975. Derivation of New Readability Formulas (Automated Readability Index, Fog Count and Flesch Reading Ease Formula) for Navy Enlisted Personnel.
- [123] Andrew J. Ko, Robert DeLine, and Gina Venolia. 2007. Information Needs in Collocated Software Development Teams. In *Proceedings of the International Conference on Software Engineering*. 344–353. <https://doi.org/10.1109/ICSE.2007.45>

- [124] Andrew J. Ko, Brad A. Myers, Michael J. Coblenz, and Htet Htet Aung. 2006. An Exploratory Study of How Developers Seek, Relate, and Collect Relevant Information During Software Maintenance Tasks. *IEEE Transactions on Software Engineering* 32, 12 (2006), 971–987. <https://doi.org/10.1109/TSE.2006.116>
- [125] Douglas Kramer. 1999. API Documentation from Source Code Comments. In *Proceedings of the International Conference on Computer Documentation*. <https://doi.org/10.1145/318372.318577>
- [126] John R Krebs, David W Stephens, William J Sutherland, et al. 1983. Perspectives in Optimal Foraging. *Perspectives in Ornithology* (1983), 165–216.
- [127] Barry M. Kroll. 1984. Writing for Readers: Three Perspectives on Audience. *College Composition and Communication* 35 (1984), 172–185. <https://doi.org/10.2307/358094>
- [128] Bill Kules and Ben Shneiderman. 2008. Users Can Change Their Web Search Tactics: Design Guidelines for Categorized Overviews. *Information Processing and Management* 44, 2 (2008), 463–484.
- [129] Sergei Kuznetsov. 2007. On Stability of a Formal Concept. *Annals of Mathematics and Artificial Intelligence* (2007). <https://doi.org/10.1007/s10472-007-9053-6>
- [130] Sergei Kuznetsov, Sergei Obiedkov, and Camille Roth. 2007. Reducing the Representation Complexity of Lattice-Based Taxonomies. In *Conceptual Structures: Knowledge Architectures for Smart Applications*. https://doi.org/10.1007/978-3-540-73681-3_18
- [131] J. Richard Landis and Gary G. Koch. 1977. The Measurement of Observer Agreement for Categorical Data. *Biometrics* 33 (1977), 159–174. <https://doi.org/10.2307/2529310>
- [132] Amy N. Langville and Carl D. Meyer. 2006. *Google’s PageRank and Beyond: The Science of Search Engine Rankings*. Princeton university press.
- [133] Thomas D. LaToza, Gina Venolia, and Robert DeLine. 2006. Maintaining Mental Models: A Study of Developer Work Habits. In *Proceedings of the International Conference on Software Engineering*. <https://doi.org/10.1145/1134285.1134355>
- [134] Joseph Lawrance, Rachel Bellamy, Margaret Burnett, and Kyle Rector. 2008. Using Information Scent to Model the Dynamic Foraging Behavior of Programmers in Maintenance Tasks. In *Proceedings of the Conference on Human Factors in Computing Systems*. 1323–1332. <https://doi.org/10.1145/1357054.1357261>
- [135] Jonathan Lazar, Jinjuan Heidi Feng, and Harry Hochheiser. 2017. Chapter 11 - Analyzing Qualitative Data. In *Research Methods in Human Computer Interaction (Second Edition)* (second edition ed.). Morgan Kaufmann, 299–327.

- [136] Jonathan Lazar, Jinjuan Heidi Feng, and Harry Hochheiser. 2017. Chapter 6 - Diaries. In *Research Methods in Human Computer Interaction (Second Edition)* (second edition ed.). Morgan Kaufmann, 135–152.
- [137] Jonathan Lazar, Jinjuan Heidi Feng, and Harry Hochheiser. 2017. Chapter 8 - Interviews and focus groups. In *Research Methods in Human Computer Interaction* (second edition ed.).
- [138] Helena Lee and Natalie Pang. 2018. Understanding the Effects of Task and Topical Knowledge in the Evaluation of Websites as Information Patch. *Journal of Documentation* 74, 1 (2018), 162–186. <https://doi.org/10.1108/JD-04-2017-0050>
- [139] Gerhard Leitner, Alexander Felfernig, Paul Blazek, Florian Reinfrank, and Gerald Ninaus. 2014. Chapter 8 - User Interfaces for Configuration Environments. In *Knowledge-Based Configuration*. Morgan Kaufmann, Boston, 89–106. <https://doi.org/10.1016/B978-0-12-415817-7.00008-6>
- [140] Dan Li. 2005. *Why Do You Blog: A Uses-and-Gratifications Inquiry into Bloggers' Motivations*. Master's thesis.
- [141] Hongwei Li, Zhenchang Xing, Xin Peng, and Wenyun Zhao. 2013. What Help do Developers Seek, When and How?. In *Proceedings of the Working Conference on Reverse Engineering*. 142–151. <https://doi.org/10.1109/WCRE.2013.6671289>
- [142] Jundong Li, Kewei Cheng, Suhang Wang, Fred Morstatter, Robert P. Trevino, Jiliang Tang, and Huan Liu. 2017. Feature Selection: A Data Perspective. *Computing Surveys (ACM)* (2017). <https://doi.org/10.1145/3136625>
- [143] Sherlock A. Licorish and Stephen G. MacDonell. 2017. Exploring Software Developers' Work Practices: Task Differences, Participation, Engagement, and Speed of Task Resolution. *Information & Management* 54, 3 (2017), 364–382. <https://doi.org/10.1016/j.im.2016.09.005>
- [144] Jay Lieberman. 1991. Chapter 3 - A Schematic Approach to User Knowledge and Software Documentation Production. In *Perspectives on Software Documentation: Inquiries and Innovations*.
- [145] Eden Litt. 2012. Knock, Knock. Who's There? The Imagined Audience. *Journal of Broadcasting & Electronic Media* 56, 3 (2012). <https://doi.org/10.1080/08838151.2012.705195>
- [146] Michael Xieyang Liu, Aniket Kittur, and Brad A. Myers. 2021. To Reuse or Not To Reuse? A Framework and System for Evaluating Summarized Knowledge. *Proceedings of the ACM on Human-Computer Interaction*, Article 166 (2021), 35 pages. <https://doi.org/10.1145/3449240>

- [147] Xinhong Liu and Reid Holmes. 2020. Exploring Developer Preferences for Visualizing External Information Within Source Code Editors. In *Proceedings of the Working Conference on Software Visualization*. 27–37. <https://doi.org/10.1109/VISSOFT51673.2020.00008>
- [148] Nicole Loorbach, Joyce Karreman, and Michael Steehouder. 2007. The Effects of Adding Motivational Elements to User Instructions. In *Proceedings of the International Professional Communication Conference*. <https://doi.org/10.1109/IPCC.2007.4464078>
- [149] Yihan Lu and I-Han Hsiao. 2017. Personalized Information Seeking Assistant (PiSA): From Programming Information Seeking to Learning. *Information Retrieval* 20 (2017), 433–455. <https://doi.org/10.1007/s10791-017-9305-y>
- [150] Niels Windfled Lund. 2007. What is Documentation? English Translation of the Classic French Text. *Journal of Documentation* (2007). <https://doi.org/10.1353/lac.2008.0003>
- [151] Renkai Ma, Xinning Gui, and Yubo Kou. 2023. Multi-Platform Content Creation: The Configuration of Creator Ecology through Platform Prioritization, Content Synchronization, and Audience Management. In *Proceedings of the Conference on Human Factors in Computing Systems*. Article 242. <https://doi.org/10.1145/3544548.3581106>
- [152] Walid Maalej and Martin P. Robillard. 2013. Patterns of Knowledge in API Reference Documentation. *IEEE Transactions on Software Engineering* 39, 9 (2013), 1264–1282. <https://doi.org/10.1109/TSE.2013.12>
- [153] Robert L. Mack, Clayton H. Lewis, and John M. Carroll. 1983. Learning to Use Word Processors: Problems and Prospects. *ACM Transactions on Information Systems* 1, 3 (1983), 254–271. <https://doi.org/10.1145/357436.357440>
- [154] Jo Mackiewicz. 2005. Use and Effect of Declarative Information in User Instructions. *Transactions on Professional Communication* 48, 1 (2005). <https://doi.org/10.1109/TPC.2004.843302>
- [155] Laura MacLeod, Margaret-Anne Storey, and Andreas Bergen. 2015. Code, Camera, Action: How Software Developers Document and Share Program Knowledge Using YouTube. In *IEEE International Conference on Program Comprehension*. <https://doi.org/10.1109/ICPC.2015.19>
- [156] Jennifer H. Maher. 2011. The Technical Communicator as Evangelist: Toward Critical and Rhetorical Literacies of Software Documentation. *Journal of Technical Writing and Communication* 41 (2011). Issue 4. <https://doi.org/10.2190/TW.41.4.d>
- [157] Taryn Marks and Avery Le. 2017. Increasing Article Findability Online: The Four Cs of Search Engine Optimization. *Law Library Journal* 109, 11 (2017).

- [158] Arthur Marques, Nick C. Bradley, and Gail C. Murphy. 2020. Characterizing Task-Relevant Information in Natural Language Software Artifacts. *Proceedings of the International Conference on Software Maintenance and Evolution* (2020), 476–487. <https://doi.org/10.1109/ICSME46990.2020.00052>
- [159] Gregory R McArthur. 1986. If Writers Can’t Program and Programmers Can’t Write, Who’s Writing User Documentation?. In *Proceedings of International Conference on Systems Documentation*. <https://doi.org/10.1145/10563.10574>
- [160] Sarah McRoberts, Elizabeth Bonsignore, Tamara Peyton, and Svetlana Yarosh. 2016. Do It for the Viewers! Audience Engagement Behaviors of Young YouTubers. In *Proceedings of the International Conference on Interaction Design and Children*. <https://doi.org/10.1145/2930674.2930676>
- [161] Edward Meade, Emma O’Keeffe, Niall Lyons, Dean Lynch, Murat Yilmaz, Ulas Gulec, Rory V. O’Connor, and Paul M. Clarke. 2019. The Changing Role of the Software Engineer. In *Systems, Software and Services Process Improvement*. 682–694. https://doi.org/10.1007/978-3-030-28005-5_53
- [162] Yevgeniy Medynskiy, Mira Dontcheva, and Steven M. Drucker. 2009. Exploring Websites Through Contextual Facets. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 2013–2022. <https://doi.org/10.1145/1518701.1519007>
- [163] Brad Mehlenbacher. 2002. Documentation: Not Yet Implemented, but Coming Soon! *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications* (2002), 527–543.
- [164] Cyrus R Mehta and Nitin R Patel. 2011. IBM SPSS Exact Tests. *Armonk, NY: IBM Corporation* (2011).
- [165] Michael Meng, Stephanie Steinhardt, and Andreas Schubert. 2018. Application Programming Interface Documentation: What Do Software Developers Want? *Journal of Technical Writing and Communication* 48 (2018), 295–330. Issue 3. <https://doi.org/10.1177/0047281617721853>
- [166] Michael Meng, Stephanie Steinhardt, and Andreas Schubert. 2019. How Developers use API Documentation: An Observation Study. In *Communication Design Quarterly Review*, Vol. 7. 40–49. <https://doi.org/10.1145/3358931.3358937>
- [167] Michael Meng, Stephanie M. Steinhardt, and Andreas Schubert. 2020. Optimizing API Documentation: Some Guidelines and Effects. In *Proceedings of the International Conference on Design of Communication*. <https://doi.org/10.1145/3380851.3416759>
- [168] Aliaksei Miniukovich, Antonella De Angeli, Simone Sulpizio, and Paola Venuti. 2017. Design Guidelines for Web Readability. In *Proceedings of the Conference on Designing Interactive Systems*. 285–296. <https://doi.org/10.1145/3064663.3064711>

- [169] Azuka Mordi and Mareike Schoop. 2020. Making It Tangible - Creating a Definition of Agile Mindset. In *Proceedings of the European Conference on Information Systems*.
- [170] Simona Motogna, Dan Mircea Suci, and Arthur-Jozsef Molnar. 2021. Agile Mindset Adoption in Student Team Projects. In *International Conference on Evaluation of Novel Approaches to Software Engineering*. https://doi.org/10.1007/978-3-030-96648-5_13
- [171] Donald M. Murray. 1982. Teaching the Other Self: The Writer's First Reader. *College Composition and Communication* 33, 2 (1982). <https://doi.org/10.2307/357621>
- [172] Sarah Nadi and Christoph Treude. 2020. Essential Sentences for Navigating Stack Overflow Answers. In *Proceedings of the International Conference on Software Analysis, Evolution and Reengineering*. 229–239. <https://doi.org/10.1109/SANER48275.2020.9054828>
- [173] Daye Nam, Andrew Macvean, Vincent Hellendoorn, Bogdan Vasilescu, and Brad Myers. 2024. Using an LLM to Help With Code Understanding. In *Proceedings of the International Conference on Software Engineering*. Article 97. <https://doi.org/10.1145/3597503.3639187>
- [174] Mathieu Nassif, Zara Horlacher, and Martin P. Robillard. 2022. Casdoc: Unobtrusive Explanations in Code Examples. In *Proceedings of the International Conference on Program Comprehension*. 631–635. <https://doi.org/10.1145/3524610.3527875>
- [175] Mathieu Nassif and Martin P. Robillard. 2021. Wikifying Software Artifacts. *Empirical Software Engineering* 26 (2021). <https://doi.org/10.1007/s10664-020-09918-4>
- [176] Mathieu Nassif and Martin P. Robillard. 2023. A Field Study of Developer Documentation Format. In *Proceedings of the Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*. 1–7. <https://doi.org/10.1145/3544549.3585767>
- [177] Mitchell J Nathan, Kenneth R Koedinger, Martha W Alibali, et al. 2001. Expert Blind Spot: When Content Knowledge Eclipses Pedagogical Content Knowledge. In *Proceedings of the International Conference on Cognitive Science*.
- [178] Oleg Nenadic and Michael Greenacre. 2007. Correspondence Analysis in R, with Two- and Three-dimensional Graphics: The ca Package. (2007). <https://doi.org/10.18637/jss.v020.i03>
- [179] Octoverse. 2021. The 2021 State of the Octoverse. <https://octoverse.github.com/2021/> Accessed December 2024.
- [180] OpenAI. Accessed 2024. ChatGPT (Large Language Model). <https://chat.openai.com/chat>
- [181] Stack Overflow. 2016. Developer Survey Results. <https://survey.stackoverflow>.

- co/2016#work-challenges-at-work Accessed December 2024.
- [182] Stack Overflow. 2017. Developer Survey Results. <https://survey.stackoverflow.co/2017#developer-profile--other-types-of-education> Accessed December 2024.
- [183] Stack Overflow. Accessed December 2024. Stack Overflow Annual Developer Survey. <https://survey.stackoverflow.co>
- [184] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. *The PageRank Citation Ranking: Bringing Order to the Web*. Technical Report. <http://ilpubs.stanford.edu:8090/422/>
- [185] Eduardo G. Q. Palmeira, Abiel Roche-Lima, and André B. de Sales. 2020. Users Preferences Regarding Types of Help: Different Contexts Comparison. In *Information Technology and Systems*. 307–314. https://doi.org/10.1007/978-3-030-40690-5_30
- [186] Cécile Paris, Keith Vander Linden, and Shijian Lu. 2002. Automated Knowledge Acquisition for Instructional Text Generation. In *Proceedings of the Annual International Conference on Computer Documentation*. <https://doi.org/10.1145/584955.584977>
- [187] Chris Parnin and Christoph Treude. 2011. Measuring API Documentation on the Web. In *Proceedings of the International Workshop on Web 2.0 for Software Engineering*. <https://doi.org/10.1145/1984701.1984706>
- [188] Chris Parnin, Christoph Treude, Lars Grammel, and Margaret-Anne Storey. 2012. Crowd Documentation: Exploring the Coverage and the Dynamics of API Discussions on Stack Overflow. *Georgia Technical Report* (2012).
- [189] Chris Parnin, Christoph Treude, and Margaret-Anne Storey. 2013. Blogging Developer Knowledge: Motivations, Challenges, and Future Directions. In *Proceedings of International Conference on Program Comprehension*. <https://doi.org/10.1109/ICPC.2013.6613850>
- [190] Aleksandra Pawlik, Judith Segal, and Marian Petre. 2012. Documentation Practices in Scientific Software Development. In *Proceedings of the International Workshop on Co-operative and Human Aspects of Software Engineering*. 113–119. <https://doi.org/10.1109/CHASE.2012.6223004>
- [191] Gayane Petrosyan, Martin P Robillard, and Renato De Mori. 2015. Discovering Information Explaining API Types Using Text Classification. In *Proceedings of the International Conference on Software Engineering*. 869–879. <https://doi.org/10.1109/ICSE.2015.97>
- [192] Vir Phoha. 1997. A Standard for Software Documentation. In *Computer*.

- [193] David Piorkowski, Scott D. Fleming, Christopher Scaffidi, Margaret Burnett, Irwin Kwan, Austin Z. Henley, Jamie Macbeth, Charles Hill, and Amber Horvath. 2015. To fix or to learn? How production bias affects developers’ information foraging during debugging. In *Proceedings of the International Conference on Software Maintenance and Evolution*. 11–20. <https://doi.org/0.1109/ICSM.2015.7332447>
- [194] Peter Pirolli and Stuart Card. 1999. Information Foraging. In *Psychological Review*, Vol. 106. 643–675. <https://doi.org/10.1037/0033-295X.106.4.643>
- [195] Peter Pirolli and Wai-tat Fu. 2003. SNIF-ACT: A Model of Information Foraging on the World Wide Web. In *Proceedings of the International Conference on User Modeling*. https://doi.org/10.1007/3-540-44963-9_8
- [196] Michael Priestley. 1999. Dynamically Assembled Documentation. In *Proceedings of the Annual International Conference on Computer Documentation*. 53–57. <https://doi.org/10.1145/318372.318556>
- [197] Uta Priss. 2006. Formal Concept Analysis in Information Science. *Annual Review of Information Science and Technology* (2006).
- [198] Daniele Procida. 2023. Diátaxis Documentation Framework. <https://diataxis.fr/>
- [199] Rick Rabiser, Michael Vierhauser, Martin Lehofer, Paul Grünbacher, and Tomi Männistö. 2014. Configuring and Generating Technical Documents. In *Knowledge-Based Configuration*. Elsevier, 241–250. <https://doi.org/10.1016/B978-0-12-415817-7.00020-7>
- [200] Sruti Srinivasa Ragavan, Sandeep Kaur Kuttal, Charles Hill, Anita Sarma, David Piorkowski, and Margaret Burnett. 2016. Foraging Among an Overabundance of Similar Variants. In *Proceedings of the Conference on Human Factors in Computing Systems*. 3509–3521. <https://doi.org/10.1145/2858036.2858469>
- [201] Marco Raglianti, Csaba Nagy, Roberto Minelli, Bin Lin, and Michele Lanza. 2023. On the Rise of Modern Software Documentation (Pearl/Brave New Idea). In *European Conference on Object-Oriented Programming*. <https://doi.org/10.4230/LIPICS.ECOP.2023.43>
- [202] Othmane Rahmaoui, Kamal Souali, and Mohammed Ouzzif. 2019. Towards an Improvement of the Software Documentation using a Traceability Approach. In *Proceedings of the Advanced Intelligent Systems for Sustainable Development*. https://doi.org/10.1007/978-3-030-11928-7_46
- [203] Pooja Rani, Suada Abukar, Nataliia Stulova, Alexandre Bergel, and Oscar Nierstrasz. 2021. Do Comments follow Commenting Conventions? A Case Study in Java and Python. In *Proceedings of the International Working Conference on Source Code Analysis and Manipulation*. 165–169. <https://doi.org/10.1109/SCAM52516.2021.00028>

- [204] Nikitha Rao, Chetan Bansal, Thomas Zimmermann, Ahmed Hassan Awadallah, and Nachiappan Nagappan. 2020. Analyzing Web Search Behavior for Software Engineering Tasks. In *Proceedings of the International Conference on Big Data*. <https://doi.org/10.1109/BigData50022.2020.9378083>
- [205] Marc Rettig. 1991. Nobody Reads Documentation. *Commun. ACM* (1991).
- [206] R. Ries. 1990. IEEE Standard for Software User Documentation. In *Proceedings of the International Conference on Professional Communication, Communication Across the Sea: North American and European Practices*.
- [207] Martin P Robillard. 2009. What Makes APIs Hard to Learn? The Answers of Developers. *IEEE Software* (2009). <https://doi.org/10.1109/MS.2009.193>
- [208] Martin P. Robillard. 2021. Turnover-induced Knowledge Loss in Practice. In *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. <https://doi.org/10.1145/3468264.3473923>
- [209] 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. 2024. Communicating Study Design Trade-offs in Software Engineering. *Transactions on Software Engineering and Methodology* (2024). <https://doi.org/10.1145/3649598>
- [210] Martin P Robillard and Robert Deline. 2011. A Field Study of API Learning Obstacles. *Empirical Software Engineering* (2011). <https://doi.org/10.1007/s10664-010-9150-8>
- [211] Martin P. Robillard, Andrian Marcus, Christoph Treude, Gabriele Bavota, Oscar Chaparro, Neil Ernst, Marco Aurélio Gerosa, Michael Godfrey, Michele Lanza, Mario Linares-Vásquez, Gail C. Murphy, Laura Moreno, David Shepherd, and Edmund Wong. 2017. On-demand Developer Documentation. In *Proceedings of the International Conference on Software Maintenance and Evolution*. 479–483. <https://doi.org/10.1109/ICSME.2017.17>
- [212] Bernard R. Robin. 2008. The Effective Uses of Digital Storytelling as a Teaching and Learning Tool. In *Handbook of Research on Teaching Literacy Through the Communicative and Visual Arts*.
- [213] Camille Roth, Sergei Obiedkov, and Derrick Kourie. 2008. Towards Concise Representation for Taxonomies of Epistemic Communities. In *Concept Lattices and Their Applications*. https://doi.org/10.1007/978-3-540-78921-5_17
- [214] Riccardo Rubei, Claudio Di Sipio, Phuong T. Nguyen, Juri Di Rocco, and Davide Di Ruscio. 2020. PostFinder: Mining Stack Overflow Posts to Support Software De-

- velopers. *Information and Software Technology* 127 (2020). <https://doi.org/10.1016/j.infsof.2020.106367>
- [215] Daniel Russo. 2021. The Agile Success Model: A Mixed-methods Study of a Large-scale Agile Transformation. *Transactions on Software Engineering and Methodologies* 30, 4 (2021). <https://doi.org/10.1145/3464938>
- [216] Richard M. Ryan and Edward L. Deci. 2020. Intrinsic and Extrinsic Motivation from a Self-determination Theory Perspective: Definitions, Theory, Practices, and Future Directions. *Contemporary Educational Psychology* 61 (2020). <https://doi.org/10.1016/j.cedpsych.2020.101860>
- [217] Caitlin Sadowski, Kathryn T. Stolee, and Sebastian Elbaum. 2015. How Developers Search for Code: A Case Study. In *Proceedings of the 10th Joint Meeting on Foundations of Software Engineering*. 191–201. <https://doi.org/10.1145/2786805.2786855>
- [218] Hani Safadi, Steven L. Jonson, and Samer Faraj. 2020. Who Contributes Knowledge? Core-periphery Tension in Online Innovation Communities. *Organization Science* 32, 3 (2020). <https://doi.org/10.1287/orsc.2020.1364>
- [219] Carol Sansone and Jessi L. Smith. 2000. The "How" of Goal Pursuit: Interest and Self-Regulation. *Psychological Inquiry* 11, 4 (2000).
- [220] Jan Schmidt. 2007. Blogging Practices: An Analytical Framework. *Journal of Computer-Mediated Communication* 12 (2007). Issue 4. <https://doi.org/10.1111/j.1083-6101.2007.00379.x>
- [221] Kurt Schneider. 2009. *Experience and Knowledge Management in Software Engineering*. Springer. <https://doi.org/10.1007/978-3-540-95880-2>
- [222] Daniel Schreck, Valentin Dallmeier, and Thomas Zimmermann. 2007. How Documentation Evolves Over Time. In *Proceedings of the Ninth International Workshop on Principles of Software Evolution: In Conjunction with the 6th ESEC/FSE Joint Meeting*. 4–10. <https://doi.org/10.1145/1294948.1294952>
- [223] Carolyn B. Seaman. 1999. Qualitative Methods in Empirical Studies of Software Engineering. *IEEE Transactions on Software Engineering* 25 (1999). Issue 4. <https://doi.org/10.1109/32.799955>
- [224] Donald Sharpe. 2015. Chi-square Test is Statistically Significant: Now What? *Practical Assessment, Research, and Evaluation* 20, 1 (2015). <https://doi.org/10.7275/tbfa-x148>
- [225] Miriam Gamoran Sherin. 2002. When Teaching Becomes Learning. *Cognition and Instruction* 20, 2 (2002). https://doi.org/10.1207/S1532690XCI2002_1

- [226] Lin Shi, Xiao Chen, Ye Yang, Hanzhi Jiang, Ziyu Jiang, Nan Niu, and Qing Wang. 2021. A first look at developers’ live chat on Gitter. In *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 391–403. <https://doi.org/10.1145/3468264.3468562>
- [227] Lin Shi, Hao Zhong, Tao Xie, and Mingshu Li. 2011. An Empirical Study on Evolution of API Documentation. In *Proceedings of the Fundamental Approaches to Software Engineering*. https://doi.org/10.1007/978-3-642-19811-3_29
- [228] Yulia Shmerlin, Irit Hadar, Doron Kliger, and Hayim Makabee. 2015. To Document or Not to Document? An Exploratory Study on Developers’ Motivation to Document Code. In *Advanced Information Systems Engineering Workshops*. https://doi.org/10.1007/978-3-319-19243-7_10
- [229] Jonathan Sillito, Gail C. Murphy, and Kris De Volder. 2006. Questions Programmers Ask during Software Evolution Tasks. In *Proceedings of the International Symposium on Foundations of Software Engineering*. Association for Computing Machinery, 23–34. <https://doi.org/10.1145/1181775.1181779>
- [230] Rodrigo F Silva, Mohammad Masudur Rahman, Carlos Eduardo Dantas, Chanchal Roy, Foutse Khomh, and Marcelo A Maia. 2021. Improved Retrieval of Programming Solutions with Code Examples Using a Multi-featured Score. *The Journal of Systems & Software* 181 (2021). <https://doi.org/10.1016/j.jss.2021.111063>
- [231] Beth Simon, Krista Davis, William G. Griswold, Michael Kelly, and Roshni Malani. 2008. Noteblogging: Taking Note Taking Public. *Proceedings of the SIGCSE technical symposium on Computer Science education* (2008). <https://doi.org/10.1145/1352322.1352278>
- [232] Ian Sommerville. 2001. Software Documentation. *Software engineering* 2 (2001), 143–154.
- [233] Peter Sprent. 2011. Fisher Exact Test. In *International Encyclopedia of Statistical Science*. Springer Berlin Heidelberg, 524–525.
- [234] Sruti Srinivasa Ragavan, Sandeep Kaur Kuttal, Charles Hill, Anita Sarma, David Piorkowski, and Margaret Burnett. 2016. Foraging Among an Overabundance of Similar Variants. In *Proceedings of the Conference on Human Factors in Computing Systems*. 3509–3521. <https://doi.org/10.1145/2858036.2858469>
- [235] Jeffrey Stylos, Andrew Faulring, Zizhuang Yang, and Brad A. Myers. 2009. Improving API Documentation Using API Usage Information. In *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing*. 119–126. <https://doi.org/10.1109/VLHCC.2009.5295283>

- [236] Yiming Su, Chengcheng Wan, Utsav Sethi, Shan Lu, Madan Musuvathi, and Suman Nath. 2023. HotGPT: How to Make Software Documentation More Useful with a Large Language Model?. In *Proceedings of the 19th Workshop on Hot Topics in Operating Systems*. 87–93. <https://doi.org/10.1145/3593856.3595910>
- [237] Mohammad Tahaei, Adam Jenkins, Kami Vaniea, and Maria Wolters. 2021. “I don’t know too much about it”: On the Security Mindsets of Computer Science students. In *Proceedings of the Socio-Technical Aspects in Security and Trust*. https://doi.org/10.1007/978-3-030-55958-8_2
- [238] Andy Takats and Nathan Brewer. 2005. Improving communication between customers and developers. In *Proceedings of the Agile Development Conference*. 243–252. <https://doi.org/10.1109/ADC.2005.30>
- [239] Henry Tang and Sarah Nadi. 2023. Evaluating Software Documentation Quality. In *Proceedings of the International Conference on Mining Software Repositories*. 67–78. <https://doi.org/10.1109/MSR59073.2023.00023>
- [240] Ruixiang Tang, Yu-Neng Chuang, and Xia Hu. 2024. The Science of Detecting LLM-Generated Text. *Communications of the ACM* 67, 4 (2024), 50–59. <https://doi.org/10.1145/3624725>
- [241] Jaime Teevan, Christine Alvarado, Mark S. Ackerman, and David R. Karger. 2004. The Perfect Search Engine is Not Enough: A Study of Orienteering Behavior in Directed Search. In *Proceedings of the Conference on Human Factors in Computing Systems*. 415–422. <https://doi.org/10.1145/985692.985745>
- [242] Jaime Teevan, Edward Cutrell, Danyel Fisher, Steven M. Drucker, Gonzalo Ramos, Paul André, and Chang Hu. 2009. Visual Snippets: Summarizing Web Pages for Search and Revisitation. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 2023–2032. <https://doi.org/10.1145/1518701.1519008>
- [243] Sunil Raj Thota, Saransh Arora, and Sandeep Gupta. 2024. AI-Driven Automated Software Documentation Generation for Enhanced Development Productivity. In *Proceedings of the International Conference on Data Science and Network Security*. 1–7. <https://doi.org/10.1109/ICDSNS62112.2024.10691221>
- [244] Rebecca Tiarks and Walid Maalej. 2014. How Does a Typical Tutorial for Mobile Development Look Like?. In *Proceedings of the Working Conference on Mining Software Repositories*. 272–281. <https://doi.org/10.1145/2597073.2597106>
- [245] Thomas Tilley, Richard Cole, Peter Becker, and Peter Eklund. 2005. A Survey of Formal Concept Analysis Support for Software Engineering Activities. In *Formal Concept Analysis: Foundations and Applications*. https://doi.org/10.1007/11528784_13
- [246] Christoph Treude and Maurício Aniche. 2018. Where Does Google Find API Docu-

- mentation?. In *Proceedings of the International Conference on Software Engineering*. 23–26. <https://doi.org/10.1145/3194793.3194796>
- [247] Christoph Treude, Martin P. Robillard, and Barthélemy Dagenais. 2015. Extracting Development Tasks to Navigate Software Documentation. *IEEE Transactions on Software Engineering* (2015). <https://doi.org/10.1109/TSE.2014.2387172>
- [248] Nicole Ummelen. 1996. The Selection and Use of Procedural and Declarative Information in Software Manuals. *Journal of Technical Writing and Communication* 26 (1996). Issue 4. <https://doi.org/10.2190/FQJ1-2W2B-C886-MRY7>
- [249] Petko Valtchev, Rokia Missaoui, and Robert Godin. 2004. Formal Concept Analysis for Knowledge Discovery and Data Mining: The New Challenges. In *Concept Lattices*. https://doi.org/10.1007/978-3-540-24651-0_30
- [250] Hams Van Der Meij and Mark Gellevij. 2004. The Four Components of a Procedure. *IEEE Transactions on Professional Communication* 47, 1 (2004), 5–14. <https://doi.org/10.1109/TPC.2004.824292>
- [251] Hans van der Meij and Jan van der Meij. 2014. A Comparison of Paper-based and Video Tutorials for Software Learning. *Computers & Education* 78 (2014), 150–159. <https://doi.org/10.1016/j.compedu.2014.06.003>
- [252] Isabel Villegas-Simón, Ona Anglada-Pujol, María Castellví Lloveras, and Mercè Oliva. 2023. “I’m Not Just a Content Creator”: Digital Cultural Communicators Dealing with Celebrity Capital and Online Communities. *International Journal of Communication* 17 (2023), 19.
- [253] Sruthi Viswanathan, Behrooz Omidvar-Tehrani, and Jean-Michel Renders. 2022. What is your current mindset?. In *Proceedings of the Conference on Human Factors in Computing Systems*. <https://doi.org/10.1145/3491102.3501912>
- [254] April Yi Wang, Andrew Head, Ashley Ge Zhang, Steve Oney, and Christopher Brooks. 2023. Colaroid: A Literate Programming Approach for Authoring Explorable Multi-stage Tutorials. In *Proceedings of the Conference on Human Factors in Computing Systems*. <https://doi.org/10.1145/3544548.3581525>
- [255] April Yi Wang, Dakuo Wang, Jaimie Drozdal, Xuye Liu, Soya Park, Steve Oney, and Christopher Brooks. 2021. What Makes a Well-documented Notebook? A Case Study of Data Scientists’ Documentation Practices in Kaggle. In *Extended Abstracts of the Conference on Human Factors in Computing Systems*. <https://doi.org/10.1145/3411763.3451617>
- [256] April Yi Wang, Dakuo Wang, Jaimie Drozdal, Michael Muller, Soya Park, Justin D. Weisz, Xuye Liu, Lingfei Wu, and Casey Dugan. 2022. Documentation Matters: Human-Centered AI System to Assist Data Science Code Documentation in Com-

- putational Notebooks. *Transactions on Computer-Human Interaction* 29, 2, Article 17 (2022). <https://doi.org/10.1145/3489465>
- [257] Wengran Wang, Archit Kwatra, James Skripchuk, Neeloy Gomes, Alexandra Milliken, Chris Martens, Tiffany Barnes, and Thomas Price. 2021. Novices’ Learning Barriers When Using Code Examples in Open-Ended Programming. In *Proceedings of the Conference on Innovation and Technology in Computer Science Education*. 394–400. <https://doi.org/10.1145/3430665.3456370>
- [258] Robert Ward. 1993. The Content and Organisation of User Documentation for Information Systems. In *Colloquium on Issues in Computer Support for Documentation and Manuals*.
- [259] Robert Watson, Mark Stamnes, Jacob Jeannot-Schroeder, and Jan H. Spyridakis. 2013. API Documentation and Software Community Values: A Survey of Open-source API Documentation. In *Proceedings of the International Conference on Design of Communication*. 165–174. <https://doi.org/10.1145/2507065.2507076>
- [260] Robert B. Watson. 2012. Development and Application of a Heuristic to Assess Trends in API Documentation. In *Proceedings of the ACM International Conference on Design of Communication*. <https://doi.org/10.1145/2379057.2379112>
- [261] Dustin J. Welbourne and Will J. Grant. 2016. Science Communication on YouTube: Factors That Affect Channel and Video Popularity. *Public Understanding of Science* 25, 6 (2016). <https://doi.org/10.1177/0963662515572068>
- [262] Ryen W. White, Joemon M. Jose, and Ian Ruthven. 2006. An Implicit Feedback Approach for Interactive Information Retrieval. *Information Processing and Management* 42, 1 (2006), 166–190.
- [263] Bruce Winterhalder. 1981. Optimal Foraging Strategies and Hunter-Gatherer Research in Anthropology: Theory and Models. (1981).
- [264] Patricia Wright. 1988. Chapter 28 - Issues of Content and Presentation in Document Design. In *Handbook of Human-Computer Interaction*. 629–652. <https://doi.org/10.1016/B978-0-444-70536-5.50033-6>
- [265] S. Paul Wright. 1992. Adjusted P-Values for Simultaneous Inference. *Biometrics* 48, 4 (1992), 1005–1013. <https://doi.org/10.2307/2532694>
- [266] Write the Docs 2024. <https://www.writethedocs.org> Accessed: May 29th, 2024.
- [267] Di Wu, Xiao-Yuan Jing, Hongyu Zhang, Yang Feng, Haowen Chen, Yuming Zhou, and Baowen Xu. 2023. Retrieving API Knowledge from Tutorials and Stack Overflow Based on Natural Language Queries. *Transactions on Software Engineering and Methodology* 32, 9 (2023). <https://doi.org/10.1145/3565799>

- [268] Xin Xia, Lingfeng Bao, David Lo, Pavneet Singh Kochhar, Ahmed E. Hassan, and Zhenchang Xing. 2017. What Do Developers Search For on the Web? *Empirical Software Engineering* 22, 6 (2017), 3149–3185. <https://doi.org/10.1007/s10664-017-9514-4>
- [269] Tao Xiao, Christoph Treude, Hideaki Hata, and Kenichi Matsumoto. 2024. DevGPT: Studying Developer-ChatGPT Conversations. In *Proceedings of the International Conference on Mining Software Repositories*. 227–230.
- [270] Iris Xie and Soohyung Joo. 2012. Factors Affecting the Selection of Search Tactics: Tasks, Knowledge, Process, and Systems. *Information Processing and Management* 48, 2 (2012), 254–270. <https://doi.org/10.1016/j.ipm.2011.08.009>
- [271] Wenjing Xie. 2016. “I am blogging...”: A Qualitative Study of Bloggers’ Motivations of Writing Blogs. In *Encyclopedia of E-Commerce Development, Implementation, and Management*. <https://doi.org/10.4018/978-1-4666-9787-4.ch142>
- [272] Yuankai Xue, Hanlin Chen, Gina R. Bai, Robert Tairas, and Yu Huang. 2024. Does ChatGPT Help With Introductory Programming? An Experiment of Students Using ChatGPT in CS1. In *Proceedings of the International Conference on Software Engineering: Software Engineering Education and Training*. 331–341. <https://doi.org/10.1145/3639474.3640076>
- [273] Ka Ping Yee, Kirsten Swearingen, Kevin Li, and Marti Hearst. 2003. Faceted Metadata for Image Search and Browsing. In *Proceedings of the Conference on Human Factors in Computing Systems*. 401–408.
- [274] Annie T.T. Ying and Martin P Robillard. 2014. Selection and Presentation Practices for Code Example Summarization. In *Proceedings of the Foundations of Software Engineering*. <https://doi.org/10.1145/2635868.2635877>
- [275] Monica Younger. 1998. Technical Documents Designed to Fit the Beginner: A Recursive Process. In *Proceedings of the 16th Annual International Conference on Computer Documentation*. 93–97. <https://doi.org/10.1145/296336.296359>
- [276] Hamed Zamani and Nick Craswell. 2020. Macaw: An Extensible Conversational Information Seeking Platform. In *Proceedings of the Conference on Research and Development in Information Retrieval*. 2193–2196. <https://doi.org/10.1145/3397271.3401415>
- [277] Junji Zhi, Vahid Garousi-Yusifoglu, Bo Sun, Golar Garousi, Shawn Shahnewaz, and Guenther Ruhe. 2015. Cost, Benefits and Quality of Software Development Documentation: A Systematic Mapping. In *Journal of Systems and Software*, Vol. 99. 175–198. <https://doi.org/10.1016/j.jss.2014.09.042>

Appendix A

Replication Package for How Programmers Find Online Learning Resources

The replication package associated with our paper “How Programmers Find Online Learning Resources” [16] is available at <https://doi.org/10.5281/zenodo.7504510> [17]. Table A.1 shows the contents of the online replication package and which documents are reproduced in this appendix.

Table A.1: Contents of the replication package [17].

Artifact	Description	Appendix Section
README.md	Details of the contents of the replication package.	
Coding Guide/ Coding Guide.pdf	Instructions followed by internal and external annotators to categorize the open codes.	
Dataset/ Belief.csv Questions.csv Preferences.csv Resources.csv Cues.csv ImpressionFactors.csv	List of codes and their annotated categories for each respective component in our data set.	
RelationConnections.csv	Connections between instances of components in our data set, identified by the relation, source instance’s code, and target instance’s code.	
Documents For Study/ Demographic Questions.txt	Text file of demographic questions for participants to complete before the study.	A.1
Diary Entry.txt	Text file of diary entry template for participants to fill in during the study.	
Questionnaire.txt	Text file of questions for participants to fill in after the study.	A.2

A.1 Demographic Questions

Below are the demographic questions that we asked participants to complete before the study began.

1. Please describe your current occupation?
2. Please describe your formal training in computing (degree, certification, etc.). Please include details such as specialization, co-op programs, date of completion, etc.
3. Please describe your programming experience. Include and briefly describe your main experiences and their duration (job, internship, personal project, major course project, etc.).
4. What is the technology that you are currently trying to learn?
5. How long have you been learning this technology?
6. Why are you learning this technology?

A.2 Post-study Questionnaire

Below are the questions that we asked participants to complete after the diary and interview study was completed.

1. What was the toughest or most frustrating part about searching for information?
2. What aspects of the tools you use for searching for information made the search easier for you?
3. Was it useful to document the steps you took to find information for subsequent searches of the same documentation? If so, in what way were they helpful? If not, what do you think would be more helpful?
4. What feature(s) would you like to assist your search for information?
5. Any other comments or feedback on the study?

Appendix B

Replication Package for Properties and Styles of Software Tutorials

The replication package associated with our paper “Properties and Styles of Software Technology Tutorials” [19] is available at <https://doi.org/10.5281/zenodo.10048532> [18]. Tables B.1 and B.2 shows the contents of the online replication package and which documents are reproduced in this appendix.

Table B.1: Contents of the replication package (executable scripts) [18].

Artifact	Description	Appendix Section
analysis_scripts/		
requirements.txt	Required python libraries to run scripts in this folder.	
map_property_names.py	Helper script to map full property names to abbreviated forms for readability of plots.	
plot_property_variations.py	Script to create the plots of resource property variations per programming language.	
statistical_analysis.R	Script to perform the ANOVA test to investigate the associations between properties across programming languages and compute and plot the correlations between properties per programming language.	
plot_properties_vs_metrics.py	Script to create the plots between each resource property and website traffic metric.	

APPENDIX B. REPLICATION PACKAGE FOR PROPERTIES AND STYLES OF SOFTWARE TUTORIALS

Table B.2: Contents of the replication package (dataset and results) [18].

Artifact	Description	Appendix Section
README.md	Details of the contents of the replication package	B.1
ResourceCollection.md	Details of the manual resource collection and filtering process.	
resource_properties/		
property_name_mappings.csv	Mapping between property name and shortform for use on diagrams	
resource_properties.csv	List of all property values of resources in our study.	
similarwebstats.csv	Traffic metrics for April to June 2023 retrieved from pro.similarweb.com.	
variations_in_property_values/ resource_properties_[programming lan- guage].pdf	Plot of resource property distributions per programming language	
anova_test_results.csv	Results of performing the anova test on resource properties across languages and across websites.	
correlations_between_properties/ significant_correlations_[programming language].pdf	Heatmap of correlation between properties of resources per programming language.	
correspondence of properties to website traf- fic/[traffic metric]/ [resource property]_vs_[traffic met- ric].pdf	Plot of design property versus website traffic metric where each point on the scatter plot maps to a resource.	
characterizing_resources/		B.2
deviation_matrix_per_resource.csv	Table of resources and their property deviations, identified distinguishing attributes, and prominent style	
fca/ FCA Intermediary Results.pdf	Details of the results of the intermediary steps used to identify Recurring Resource Styles for resources per programming language.	
Recurring Resource Styles (Table VI).pdf	Details of Recurring Resource Styles for resources per programming language.	
formal_context_[programming lan- guage].csv	Formal context for resources and attributes per programming language.	

Recurring Resource Styles (Table VI).pdf — Table VI refers to the table in the published article [19], not to a Table VI in this manuscript.

B.1 Resource Collection

We followed a manual web search to identify and retrieve popular tutorial resources for each programming language (PL) for the study:

1. Manually collect the non-advertisement results on the first three pages of the searches of ‘<PL> tutorial’, ‘<PL> programming tutorial’, and ‘<PL> development tutorial’ in the search engine DuckDuckGo (in Google Chrome browser in incognito mode) using default search configurations.
2. Identify the common search results from these three sets of search results. Include only results in text format.
3. Filter out search results that are not a part of a multi-page comprehensive learning tutorial for the programming language. For example, we discarded the search result “Learn Java” on the Stackify website because it was an index of Java tutorials, rather than a page in a comprehensive tutorial for Java.
4. Identify the table of contents of the tutorial in which the search result corresponds to a single page. Retrieve every web page, i.e. resource, of the tutorial.

Resource filtering

Discard resources that:

- do not provide information about the corresponding programming language,
- or follow a recognizable non-tutorial format, e.g. reference documentation, Q&A, etc.



































B.2 Recurring Resource Styles

Details of the data preprocessing and running of FCA to identify *resource styles* for each programming language.

	Java	C#	Python	Javascript	Typescript
Number of attributes	30	30	30	30	30
Number of attributes selected after: Zero-variance thresholding	24	24	24	24	24
Total Number of formal concepts	4493	3822	4178	1944	1033
Number of formal concepts after concept selection	1245	730	916	280	61
Number of frequent, stable concepts	8	14	13	10	15
Number of maximal frequent, stable concepts	6	14	13	9	11

APPENDIX B. REPLICATION PACKAGE FOR PROPERTIES AND STYLES OF SOFTWARE TUTORIALS

Recurring Resource Styles in our data set for C#, Javascript, and Typescript.

Resource Style	Stability	Support	NR	NN	Websites	LNRW	NRWM
C#							Educa
Contiguous With-short-paragraphs Code-heavy With-short-snippets	1.0	0.032	22	19		10 (TutorialsTeacher)	1
Fragmented Hierarchical Code-light With-long-snippets	0.999	0.029	20	11		20 (Dot.NetTutorials)	0
Short Contiguous Flat Code-heavy	0.999	0.032	22	16		11 (NetInformations)	0
Fragmented Text-light Table-heavy With-long-tables	0.999	0.028	19	14		11 (JavaTPoint)	3
Long Hierarchical Code-light With-long-snippets	0.999	0.031	21	12		20 (Dot.NetTutorials)	1
With-long-paragraphs Image-heavy Link-heavy Cross-linked	0.999	0.034	23	3		21 (Educa)	21
Short Fragmented Text-light Code-heavy	0.998	0.029	20	2		20 (JavaTPoint)	0
With-long-snippets Image-heavy Link-heavy Cross-linked	0.998	0.031	21	6		18 (Educa)	18
Image-heavy Non-task-oriented Link-heavy Cross-linked	0.996	0.031	21	5		21 (Educa)	21
With-long-paragraphs With-long-snippets Image-heavy Link-heavy	0.996	0.029	20	7		18 (Educa)	18
Short Text-light Code-heavy Topic-heavy	0.995	0.029	20	8		7 (JavaTPoint)	0
Code-light With-short-snippets Table-heavy With-long-tables	0.993	0.028	19	16		7 (JavaTPoint)	3
Short Text-light Code-heavy Non-task-oriented	0.992	0.035	24	11		21 (JavaTPoint)	0
With-long-paragraphs Non-task-oriented Link-heavy Cross-linked	0.992	0.029	20	7		19 (Educa)	1
Javascript							JavaTPoint
Long Contiguous Text-heavy With-external-links	1.0	0.056	19	13		19 (Info)	0
Long Contiguous Text-heavy Non-task-oriented	0.994	0.044	15	9		15 (Info)	0
With-short-paragraphs With-short-snippets Table-heavy With-long-tables	0.989	0.047	16	4		8 (JavaTPoint)	8
Fragmented Text-light Table-heavy With-long-tables	0.981	0.056	19	7		13 (JavaTPoint)	13
Fragmented Code-light With-short-snippets Table-heavy With-long-tables	0.981	0.05	17	1		11 (JavaTPoint)	11
Short Fragmented Code-light With-short-snippets	0.971	0.047	16	0		6 (Mozilla, JavaTPoint)	6
With-short-snippets Table-heavy With-long-tables Non-task-oriented	0.967	0.044	15	2		9 (JavaTPoint)	9
Fragmented With-long-paragraphs Code-light With-short-snippets Link-heavy	0.962	0.044	15	4		8 (JavaTPoint)	8
Short With-short-snippets Table-heavy With-long-tables	0.93	0.047	16	0		11 (JavaTPoint)	11
Typescript							Tutorial
Short Fragmented Text-light Task-oriented	0.091	0.058	10	1		9 (W3SchoolsBlog)	0
Short Fragmented Text-light With-long-paragraphs	0.838	0.058	10	1		8 (W3SchoolsBlog)	0
Short Fragmented Text-light Topic-heavy	0.838	0.052	9	1		8 (W3SchoolsBlog)	0
Contiguous Text-heavy With-short-paragraphs Link-heavy Cross-linked	0.836	0.041	7	3		7 (TypescriptTutorial)	7
Fragmented Text-light Code-light Table-heavy With-long-tables	0.797	0.052	9	1		6 (W3SchoolsBlog)	0
Fragmented Text-light With-long-paragraphs Task-oriented	0.785	0.052	9	1		6 (W3SchoolsBlog)	0
Fragmented Text-light With-long-snippets Task-oriented	0.781	0.041	7	1		7 (W3SchoolsBlog)	0
Contiguous Code-heavy Link-heavy Cross-linked	0.773	0.041	7	6		5 (TypescriptTutorial)	5
Text-heavy With-short-snippets Link-heavy Cross-linked	0.719	0.041	7	3		7 (TypescriptTutorial)	7
With-long-paragraphs Code-light Table-heavy With-long-tables	0.691	0.047	8	2		4 (JavaTPoint)	0
Short Fragmented With-long-paragraphs Code-light	0.691	0.047	8	2		6 (W3SchoolsBlog)	0

NR — Number of Resources of the resource style,

NN — Number of Non-overlapping resources with other resource styles,

LNRW — Largest Number of Resources of the resource style from a single Webite,

NRWM — Name and Number of Resources of the resource style from the Webite containing the Maximum resources for the programming language (see Table 4.1)

Appendix C

Replication Package for The Documentor Mindset

The replication package associated with our paper “The Software Documentor Mindset” [21] is available at <https://doi.org/10.5281/zenodo.14416777> [20]. Table C.1 shows the contents of the online replication package and which documents are reproduced in this appendix.

Table C.1: Contents of the replication package [20].

Artifact	Description	Appendix Section
README.md	Details of the contents of the replication package.	C.1
Interview Guide.md	The complete set of questions used as a guide to conduct semi-structured interviews with documentation contributors.	
DimensionsAndConsiderations.xlsx	The set of considerations across the three dimensions (separated into three sheets), and their corresponding open codes from analysing the interviews.	
Mindsets (Table 5).xlsx	The complete mapping of mindsets and their corresponding considerations across dimensions, for each informant. This is the complete version of Table 5 in the accompanying article.	C.2
Validation questionnaire.md	The validation questionnaire provided to informants for member checking.	

C.1 Interview Guide

Guide of questions used to conduct semi-structured interviews with documentation contributors.

1. What is your programming experience?
2. What was your journey to begin contributing documentation?
<Select two examples of documentation created by the interviewee>

3. How did you decide to cover this topic?
4. How did you go about creating the documentation?
 - What preparation was involved?
 - How did you go about designing the resources?
 - What was your procedure to create documentation?
 - What tools did you use to create documentation?
5. How long did it take you to create this documentation? Would you say this is typical?
6. How do you weigh the cost versus the benefit of contributing documentation for you?
7. Do you promote your documentation, and if so, how?
8. Has your documentation been unethically reused? If so, how did you handle the situation?
9. What do you think about the impact of ChatGPT and similar artificial intelligence (AI) on your documentation and your workflow?
10. Do you have any insights that I didn't ask about that you think would be worth sharing?

C.2 Validation Questionnaire

Based on interviews with 26 documentation contributors (referred to as documentors), such as yourself, we identified five mindsets that documentors have when creating and contributing documentation. Each mindset describes an important factor that a documentor thinks about during the documentation contribution process. Documentors may have multiple mindsets during the documentation contribution process, and may even have more than one mindset at the same time.

We would like your input on how well these mindsets resonate with your experience. Below is the description of each documentor mindset, followed by a few questions related to your experience of that mindset, as applicable. We really appreciate your input.

Documentation Contributor Information

We ask only for your name and email so we can ensure that the responses are only from interviewees of the original study. Your response to this form will be anonymized for the study.

1. Please fill in your name that you used for correspondence with Deeksha for the documentation contribution interview:
2. Please fill in the email that you used for correspondence with Deeksha for the documentation contribution interview:

Personal Development Mindset

This mindset focuses on how the documentation contributed by the documentor can be used to improve the documentor's own knowledge and skills, and obtain professional opportunities.

3. To what extent do you agree with the following statement?

I have experienced this mindset while contributing documentation.

- Strongly Disagree
- Disagree
- Agree
- Strongly Agree
- Unsure

4. If you have experienced this mindset, please describe how you experienced this mindset. If you have not, please type "N/A".

Prioritizing Personal Boundaries Mindset

This mindset focuses on how the documentation can be created considering the documentor's own needs and interests, and within the documentor's time and resource constraints.

5. To what extent do you agree with the following statement?

I have experienced this mindset while contributing documentation.

- Strongly Disagree
- Disagree
- Agree
- Strongly Agree
- Unsure

6. If you have experienced this mindset, please describe how you experienced this mindset. If you have not, please type "N/A".

Novelty and Value Addition Mindset

This mindset focuses on creating documentation that is different from existing online documentation.

7. To what extent do you agree with the following statement?

I have experienced this mindset while contributing documentation.

- Strongly Disagree
- Disagree
- Agree
- Strongly Agree
- Unsure

8. If you have experienced this mindset, please describe how you experienced this mindset.
If you have not, please type “N/A”.

Growth and Visibility Mindset

This mindset focuses on pursuing greater visibility of the documentation from audiences and search engines.

9. To what extent do you agree with the following statement?

I have experienced this mindset while contributing documentation.

- Strongly Disagree
- Disagree
- Agree
- Strongly Agree
- Unsure

10. If you have experienced this mindset, please describe how you experienced this mindset.
If you have not, please type “N/A”.

Content-oriented Mindset

This mindset focuses on allowing the environment and circumstances, as well as the technical information itself, to guide the creation of the documentation.

11. To what extent do you agree with the following statement?

I have experienced this mindset while contributing documentation.

- Strongly Disagree
- Disagree

- Agree
- Strongly Agree
- Unsure

12. If you have experienced this mindset, please describe how you experienced this mindset. If you have not, please type “N/A”.

Additional Comments

13. Were there any other mindsets that you experienced that we did not ask about?
14. Do you have any additional comments about the five mindsets?

Appendix D

Replication Package for How Programmers Interact with Multimodal Documentation

This appendix contains supplementary material associated with our paper “How Programmers Interact With Multimodal Documentation” [23]. Table D.1 shows the contents of this appendix.

Table D.1: Contents of Appendix D.

Artifact	Description	Appendix Section
Fisher’s test results	Results of running the 16 Fisher’s exact test on survey responses between Dimension A and Dimension B per Filter.	D.2
Contingency tables	The contingency tables for the statistically significant results not present in the main paper.	
Tasks per topic	List of programming tasks for each topic.	

D.1 Tasks per Topic

Regular Expressions

Assume that you are building a registration page for a web application in Java. You need to validate the user’s input, and retrieve and update their profile with the accepted information. To do this, you need to use regular expressions [...]

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?

Inheritance

Assume that you are building the account page for different users on an online store in Java. You need to handle account details for different types of users, i.e. the store owners and shoppers. To do this, you need to use inheritance [...]

Conceptual

You need to create a new class. What is the difference between instantiating a class as final, versus instantiating all its fields and methods as final?

How-to

To access the online store, storeowners and shoppers must create an account, by providing a standard set of information including their name. You create a common Account class, and two subclasses: OwnerAccount and ShopperAccount. A ShopperAccount contains a ShoppingCart which manages the actual shopping experience.

Write the basic class declaration for ShopperAccount, such that it fulfils the above requirements, assuming the classes Account, OwnerAccount, and ShoppingCart exist.

Debugging

For April Fools Day, you are asked to display all shoppers' names as "AprilFool". The Account class has the following method to return the name in an account:

```
public String getName() {  
    return this.aName;  
}
```

You write the following code in the class `ShopperAccount`:

```
public String getName(Boolean isAprilFools) {  
    if(isAprilFools) {  
        return "AprilFool";  
    }  
}
```

However, when you call this `getName` method, you still get the actual username instead of “AprilFool”. Why is this happening, and how can you fix this problem?

Exception Handling

Assume that you are building an online store in Java. You need to handle invalid behaviour in the software. To do this, you need to use exception handling [...]

Conceptual

You are debating whether to use a regular try/catch block, but also learn about the try-with-resource block. What is the difference between both these types of blocks?

How-to

A shopper’s account can be identified using their email ID. When a user tries to log in to the shop, your code must call the existing method

```
Account getAccount(String pEmail)
```

that retrieves the account for the entered email ID. If the method does not find an email, it throws a `NoSuchEmailException`. Write the code to call `getAccount`, and print out the message “Email Not Found” to the user, if their email is not found. Note: Please assume that `Account` and `NoSuchEmailException` exist.

Debugging

For the store’s 10 year celebrations, shoppers can upload their social media website to enter a raffle. You develop this simple code to parse the website:

```
try {  
    // Tries to open the URL  
}  
catch (IOException e) {
```

```
// Do nothing
}
catch (MalformedURLException e) {
    // Do nothing
}
```

What is the issue with this code and how can it be fixed?

D.2 Fisher’s Test Results

Results of the Fisher’s Exact Test between Modality Rating and Dimension, for each of the Filter criteria. Rows highlighted in green indicate statistically significant tests.

Dimension A	Dimension B	Filter	Pvalue	Adjusted p-value	Result
Rating	Task type	Text content	1.00E-05	1.60E-04	Statistically significant
Rating	Task type	Regular code example	1.50E-05	2.40E-04	Statistically significant
Rating	Task type	Summarized code example	1.93E-01	3.08E+00	Not statistically significant
Rating	Task type	Annotated code example	3.77E-01	6.03E+00	Not statistically significant
Rating	Task type	Table	1.81E-01	2.89E+00	Not statistically significant
Rating	Topic	Text content	4.66E-01	7.45E+00	Not statistically significant
Rating	Topic	Regular code example	3.59E-02	5.74E-01	Not statistically significant
Rating	Topic	Summarized code example	2.98E-02	4.76E-01	Not statistically significant
Rating	Topic	Annotated code example	2.71E-01	4.34E+00	Not statistically significant
Rating	Topic	Table	5.00E-06	8.00E-05	Statistically significant
Rating	Modality	Conceptual	1.50E-05	2.40E-04	Statistically significant
Rating	Modality	HowTo	5.00E-06	8.00E-05	Statistically significant
Rating	Modality	Debug	2.04E-02	3.27E-01	Not statistically significant
Rating	Modality	Regular expressions	2.45E-02	3.92E-01	Not statistically significant
Rating	Modality	Inheritance	4.97E-03	7.96E-02	Not statistically significant
Rating	Modality	Exception handling	3.00E-05	4.80E-04	Statistically significant

D.3 Contingency Tables

Task Type	Rating				
	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
Conceptual	3	4	4	10	35
HowTo	10	16	8	10	12
Debug	15	5	6	16	14

(a) Text content (adjusted p-value = $1.6e-4$)

Task Type	Rating				
	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
Conceptual	4	16	7	16	13
HowTo	5	5	2	10	34
Debug	14	11	9	14	8

(b) Regular code example (adjusted p-value = $2.4e-4$)

Modality	Rating				
	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	10	10	3	15	25
Regular code example	6	13	8	17	19
Summarized code example	11	28	7	5	12
Annotated code example	11	28	5	5	14
Table	12	33	4	5	9

(c) Exception handling (adjusted p-value = $4.8e-4$)



Adjusted Standardized Residuals and contingency tables between Task Type and Rating for Text content and Regular code examples, as well as between Modality and Rating for Exception handling. Note that the labels refer to modalities as “features” (see Section 6.1.2).