Properties and Styles of Software Technology Tutorials

Deeksha M. Arya, Jin L.C. Guo, Martin P. Robillard

***Abstract*—A large number of tutorials for popular software development technologies are available online, and those about the same technology vary widely in their presentation. We studied the design of tutorials in the software documentation landscape for five popular programming languages: Java, C#, Python, Javascript, and Typescript. We investigated the extent to which tutorial pages, i.e. *resources*, differ and report statistics of variations in resource properties. We developed a framework for characterizing resources based on their *distinguishing attributes*, i.e. properties that vary widely for the resource, relative to other resources. Additionally, we propose that a resource can be represented by its *resource style*, i.e. the combination of its distinguishing attributes. We discuss three techniques for characterizing resources based on our framework, to capture notable and relevant content and presentation properties of tutorial pages. We apply these techniques on a data set of 2551 resources to validate that our framework identifies valid and interpretable styles. We contribute this framework for reasoning about the design of resources in the online software documentation landscape.**

***Index Terms*—Software documentation, software tutorials, documentation design, tutorial properties, documentation search**

I. Introduction

SOFTWARE development technologies are usually accompanied by tutorials that provide information about the technology, including how to install, run, debug, navigate, and use it. These online tutorials may be released by technology creators or third-parties. Websites such as W3schools[(a)] are dedicated to providing resources for different technologies. In addition, any user may create and release resources, for example, in the form of a blog post [1], or as part of a community repository.[(b)] Although standards [2]–[4] and guidelines [5], [6] exist for software documentation, they are not enforced and seldom used in practice.

Thus, tutorial 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 [7]. Additionally, they must consider the consequences of these design choices [8]. Knowledge of how existing documentation is designed can support the *systematic* creation of new resources.

DM. Arya, JLC. Guo, and MP. Robillard are with McGill University.

E-mail: deeksha.arya@mail.mcgill.ca, jguo@cs.mcgill.ca, robillard@acm.org

Manuscript received April 2023

With so many variations in the style and content of available tutorials [9], programmers seeking information face a number of choices, and use *cues* to make decisions about resources to access [10]–[13]. Common 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 [14], [15], i.e. searching with content-related categories, and providing estimated time cost of search results [16]. Still, there remains ambiguity about the available design space in terms of the content and presentation of technology tutorials.

We investigated the design of programming tutorials in the current documentation landscape. We seek to answer *to what extent do software technology tutorials vary in their properties?* Based on observations of our investigation into these properties, we explore the question *how can we systematically reason about the design of software technology tutorials?* The study follows a data mining research method to extract and analyse design properties of technology tutorials. We focus on tutorials for Java, C#, Python, Javascript, and Typescript, and treat 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 propose 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*. We motivate these techniques, and discuss our observations of applying these techniques on our data set.

Our framework for characterizing resources provides insight to resource creators about the design of resources for software development technologies. Creators can use this information to make decisions about whether to adhere to existing styles, or innovate with new resource styles to fulfil a particular need. Our observations also provide a means to help resource seekers systematically identify pertinent properties when comparing resources for the same technology.

Replication Package: The details of our resource collection process, the extracted property data, and the results of the analysis are available in our online appendix.[(c)]

[(a)]https://www.w3schools.com

[(b)]E.g. http://www.cplusplus.com

[(c)]https://doi.org/10.5281/zenodo.10048532

II. Data Collection

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

A. Resource Collection

We identified popular tutorials for each software technology through a manual online web search using the search engine DuckDuckGo.[e] 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.[f] 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.[g] 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 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 obfuscated HTML,[h][1][2] 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 I.

B. 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 1).[i] Table II 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).

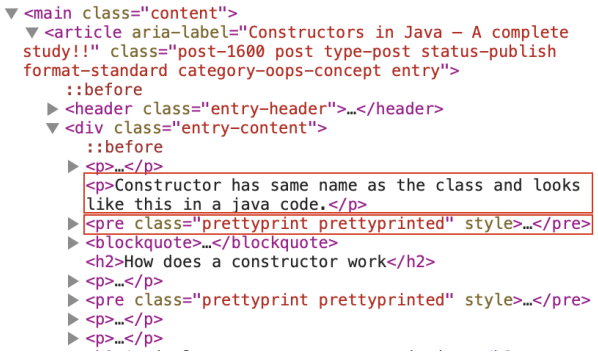


Fig. 1: Example of identified blocks for the BeginnersBook resource "Constructors in Java".[4] We manually identified that the `article` tag inside the `main` tag with `class` "content", contains the main content of the page. We treated each of these elements, such as the `<p>` and `<pre>` elements boxed in red, as an individual block.

To identify topics mentioned in resources, we used the JSI Wikifier,[j] 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 [23]. Since we focus 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 [24], 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* [22] 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 [25] such as "Output:".[5]

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 III.

III. Resource Properties

To answer our first research question *to what extent do software technology tutorials vary in their 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.

[d] https://octoverse.github.com/#top-languages-over-the-years

[e] Despite video-based learners reported to have higher success rates than text-based learners [17], the majority of results from a standard search engine are textual documents [18]. Thus, we focused our study on text tutorials.

[f] 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.

[g] The exact steps of the resource identification and filtering process are available in the replication package.

[h] Links to resources are present in the section Resource References.

[i] We used Python v3.9 and the library `beautifulsoup4` to parse HTML.

[j] https://jsi-eubusinessgraph.github.io/jsi-wikifier-api/

TABLE I: Details about the programming language and host website of the resources studied. The last three columns are estimations of the website traffic from pro.similarweb.com for the period Apr.-Jun. 2023.

	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**			

TABLE II: 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 [12].
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 III).
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 [19]. 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 [20] and length of code snippets [21] 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 [8].
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 [22].
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 [8], [19].

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

TABLE III: 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 `h1` and `h3` 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	$\frac{\text{\# of links to outside the tutorial}}{\text{total number of links}}$

A. Variations in Property Values

Figure 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 [26], 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 component[6] or describe *how* to write code to achieve a task.[7] A total of 44% (1120) of the resources contain images, the type of element preferred by computer science students and professionals in information resources [12]. Resources use images to describe installation information,[8] architecture and modelling,[9] or to annotate the code with descriptions.[10]

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 III, 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 [27], 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.[11]

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 [22], C# resources might not have instruction-like content.[12]

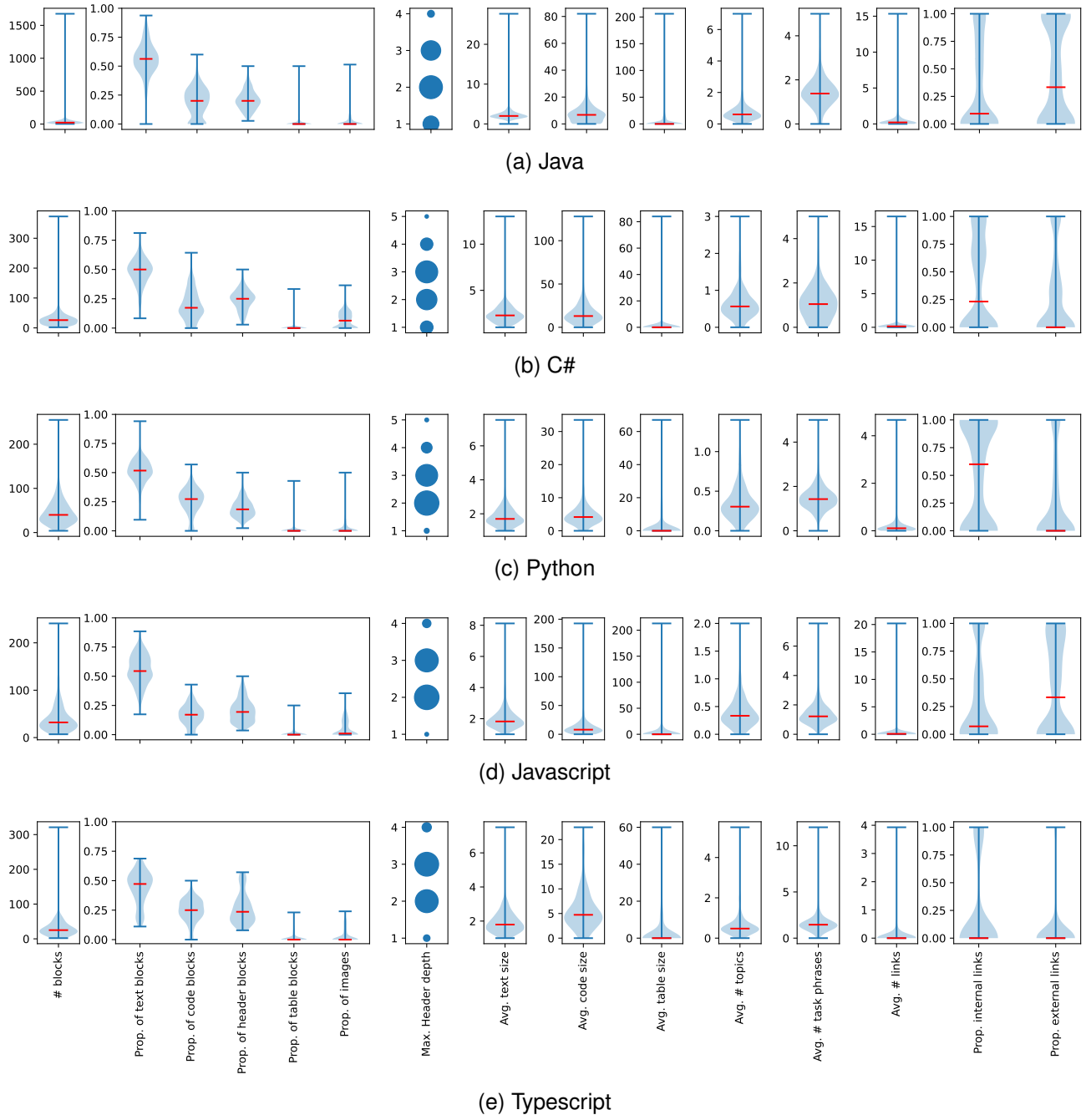
The last group of two violin plots in Figure 2 shows the proportion of the internal and external links in the resources. The distribution of the proportion of links shows greater density towards the top and/or bottom in the plots for all the languages. This indicates that, overall for a language, resources 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 [28] one-way ANOVA tests [29] ($\alpha = 0.05/15 = 0.003$ for each test). For the thirteen non-table related properties, we reject the null hypothesis, indicating that the grouping of resources by programming language 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 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 [30]; our findings suggest that further investigation into website-specific and language-specific tutorial design is worthwhile.

B. 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



(a) Java

(b) C#

(c) Python

(d) Javascript

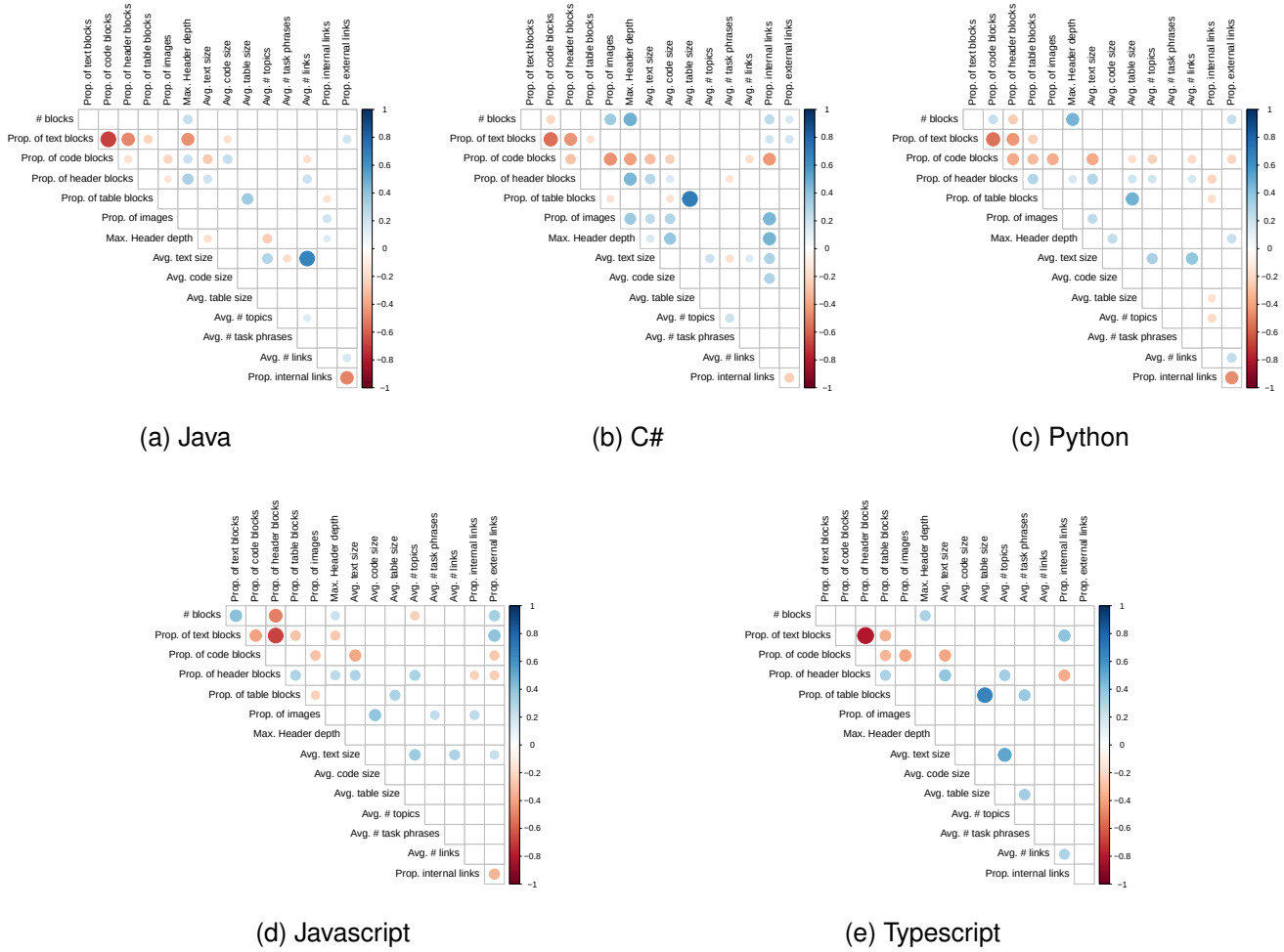
(e) Typescript

Fig. 2: Variation of resource properties by programming language. The red line indicates the median of the distribution.

programming languages), which introduces Type-I errors. To mitigate the errors, we applied Bonferroni correction [28]: $\alpha = 0.05/525 = \ 9.5 \times 10^{-5}$. Figure 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.[13]

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.[14] 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



(a) Java (b) C# (c) Python

(d) Javascript (e) Typescript

Fig. 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.

resources for detailed discussions of subtopics.[15]

C. Correspondence of Properties to Website Traffic

Software developers rate the inclusion of code examples and explanations as very important in API documentation [31], [32]. 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 I). Figure 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

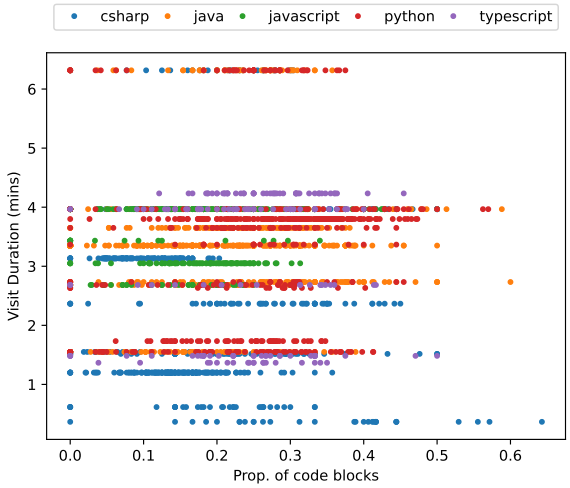


Fig. 4: Distribution of the proportion of code blocks (from Table III), against the average number of minutes per visit for the corresponding website.

documentation, which may be backed by some beliefs based on their prior resource seeking experience [13].

D. 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 [12]. 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 [8] should not be considered independently. Rather, designing a resource requires deliberation about how different aspects of the resource complement one another.

IV. Characterizing Resources

We answer 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. For example, TutorialKart's "TypeScript switch - Examples" resource[16] is comparatively short, has longer code snippets, and covers fewer topics than other Typescript tutorials in our data set.

To identify these deviations, we use quartiles of the property distribution across resources for the same programming language. We normalize each property against the maximum value of that property across resources such that all property values lay in the unit interval. We use 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 create two binary attributes: $less\_P_r$ and $more\_P_r$. We assign $less\_P_r$ 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 refer 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 IV.

For every attribute, we determine 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*.

TABLE IV: Mapping of *Less* or *More* of a property (from Table III) to an attribute of a resource.

Property	Inference of *Less*	Inference of *More*
# 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

Our observations from Section III 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 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* act 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.

A. 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 [10], [33] or cues [13] from meta information to find the information they need, however this may be implicit. Instead, a concise and explicit elicitation

TABLE V: Distinguishing attributes (d.a.) in the *prominent styles* (n=3) for all resources. The number in parentheses refers to the proportion of resources in that language that contains the corresponding attribute(s) in its prominent style.

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)

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 emails[17] has five distinguishing attributes. However, with n=3, its prominent style is identified as *Code-light, With-long-snippets, Task-oriented* because these attributes have the largest deviation for the resource.

Observations: Table V shows the most frequent distinguishing attributes in the prominent styles for each programming language.[(k)] 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 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 2e).

B. 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 [8]. To make informed choices about the content and organization of a resource, it is useful to assess existing resources [34]. 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 [35], [36]. Subsequently, creators can choose to follow existing recurring styles or identify and fill relevant design gaps.

Identification: We use Formal Concept Analysis (FCA) [37] 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 IV)[(l)] known as the *formal context*,[(m)] 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 [39] to select attributes[(n)] with a non-zero variance. This method is based on the notion that features with the same value for all data provide 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[(o)] concept selection [45] to identify important co-occurring attributes for resources. Our results from the correlation analysis (see Section III-B) 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 [46].

[(k)]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 V, the frequencies of the attributes are less than 0.25 because the prominent style for each resource is a subset of its distinguishing attributes.

[(l)]The procedure we used to convert the resource properties to binary polarized attributes is an FCA technique known as conceptual scaling [38].

[(m)]We used Python's `concepts` API to build the formal context.

[(n)]Prior attribute selection techniques are text-based [40], [41] and thus not applicable to our data. Others leveraging the distribution of objects [42] or distribution of attributes [43], rely on the size of the data set which would be biased by our attribute scaling technique. We also explored Correspondence Analysis [44], but found no clear representative dimensions.

[(o)]Note: context of our research, not formal context.

TABLE VI: 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 (JavaTPoint)	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 (BeginnersBook)	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 (BeginnersBook)	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 I)

Stability is a measure of cohesion [47]: "A concept is stable if its intent does not depend much on each particular object of the extent." [48] To calculate stability of each concept, we use the algorithm presented by Roth et al. [49] 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,[(p)] as *frequent, stable concepts* [46]. 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 VI shows the styles identified for Java and Python.[(q)] Every row provides information about one recurring style. The first column of the table indicates the combinations of attributes (from Table IV) 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.[18]
- *Short Fragmented Code-light With-short-snippets* (Python and Javascript) creates small sections to address focused information, e.g. installation related pages.[19]
- *Fragmented Text-light Table-heavy With-long-tables* (C# and Javascript) delegates some information to a table.[20]

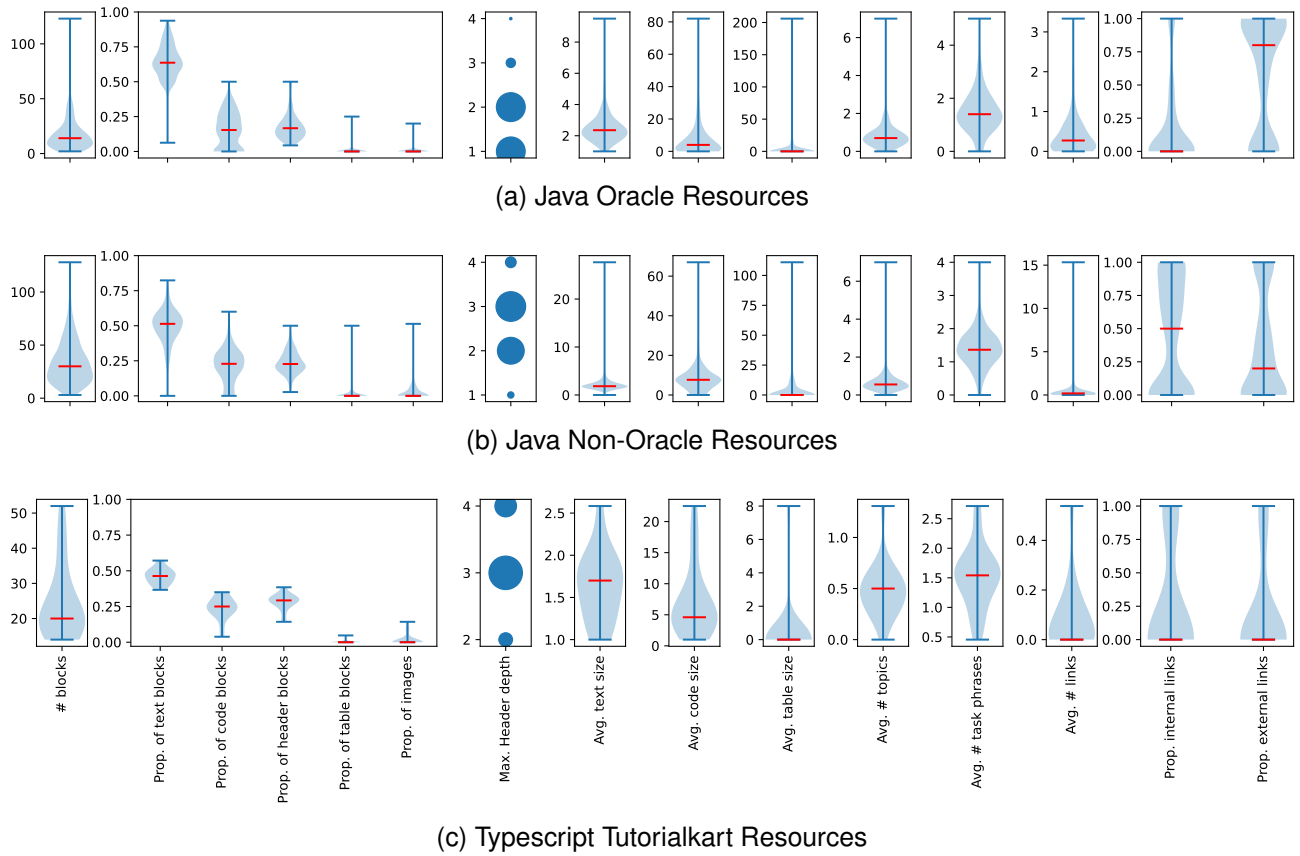
Although all the styles have a stability between 0.69 and 1, they occur in at maximum 7.5% of the resources for each language. This uniqueness among styles indicates that there is no natural progression towards a small set of common and distinct recurring styles, a possible consequence of the many design decisions taken during resource creation [8].

Whereas Python recurring styles are distributed across websites (column labelled 'Websites' in Table VI), 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 5a and 5b show the distributions of properties for Java Oracle and non-Oracle resources.[(r)] 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, corresponding to the first and fourth quartiles of the distribution. This abnormal behaviour is captured during

[(p)] Jay et al. [46] 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.

[(q)] Our results of the intermediary steps and the recurring styles for other programming languages are available in our replication package.

[(r)] We removed the abnormally long resource TutorialsPoint Java Quick Guide[21] to make a comparison.



(a) Java Oracle Resources

(b) Java Non-Oracle Resources

(c) Typescript Tutorialkart Resources

Fig. 5: Variations of extracted properties in Java Oracle, Java not-Oracle, and Typescript TutorialKart resources. The red line indicates the median of the distribution.

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 5c and Figure 2e), and thus do not have co-occurring distinguishing attributes that are captured by the resource styles.

C. 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 [13], [50] 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.[22] 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.[23] 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.

D. Discussion

The proposed framework for characterizing resources based on combinations of distinguishing attributes supports a systematic way to reason about resource design. As a result, the framework supports the comparison of multiple resources based on their design. For example, both the C# GeeksForGeeks resource on the `switch` statement[24] and the C# TutorialsTeacher resource on the same topic[25] 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 [51]. Our framework to characterize resources by their context-specific styles can be leveraged to inform the creation of flexible tutorials.

V. 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 resource[26] 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 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.[27] 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`[(s)] in the table in the Java Official resource "Object Ordering".[28] 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.[29] Pursuing perfect property extraction is time-intensive. In line with our goal to determine how properties can be used to characterize resources, we focus our research effort towards investigating how the properties co-occur. 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 [52]. We supplement the analysis with the characterization of resources based on property co-occurrences.

We retrieved traffic-related metrics (see Table I) 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.[(t)] 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.

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 parallely 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.

VI. Related Work

Our work is related to prior literature on the design of software documentation and patterns in documentation. We also discuss previous work that applies Formal Concept Analysis, the technique which we rely on to identify *recurring styles*.

A. Design of Software Documentation

Much prior research has concentrated on the information content of software documents [25], [53] and the style of the information presented [22], [54]. 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 [55].

Tiarks and Maalej performed an exploratory study of 1274 tutorials on Android, Apple iOS, and Windows Phone OS

[(s)]http://en.wikipedia.org/wiki/Path_(computing)

[(t)]We identified the number of mentions of the resource URLS in Stack Overflow answers present in the Stack Exchange Data Dump of June 2023.

to understand the nuances in mobile app development tutorials [9]. In prior work, we studied the design of three Android tutorials from different sources [8]. In both studies, the researchers reported how tutorials varied in structure and content, with the latter study providing a set of guidelines for thinking about the consequences of different design decisions. 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 [56].

Tang and Nadi developed and evaluated a tool to summarize nine metrics related to the quality of documentation including readability and ease of use [57]. Despite methods to evaluate the quality of documentation [58], there is no consensus about how documentation should be designed. The Diátaxis documentation framework consists of four structural modes, *tutorials*, *How-Tos*, *reference*, and *explanation* and provides guidelines for the overall content and styling of each mode [6]. 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 [53]. Although prior work elicits design implications based on difficulties with learning software [26] and guidelines to design documentation [59], these studies focus on documentation for Application Programming Interfaces (APIs). In this work, we use a semi-automated, data-driven approach to characterize tutorials based on their design.

B. Patterns in Documentation

Dagenais and Robillard defined *documentation patterns* as *coherent sets of code elements that are documented together* [60]. 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 *API information* providing sentences in a sample of tutorials for Java and Python could be replaced by their API reference documentation counterparts [61]. 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 [62], [63]. Butler et al. proposed a *reuse case* which documents the reuse of a framework [64]. 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, we propose the characterization of resources based on their *styles* and support the grouping of resources based on recurring styles.

C. 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 IV) [65]. In the context of software, FCA has been used to model common and uncommon features of different software product variants [66]–[69]. Prior work has investigated the ability to characterize code by the relations between classes, to support code analysis [70] and to identify design patterns [71]. As a result of literature surveys conducted by Tilley et al. [72] and Ferré et al. [73], and an investigation into FCA for data mining by Valtchev et al. [74], the authors emphasise the potential of FCA for knowledge discovery in a variety of domains, including software engineering. We leverage FCA to elicit commonly co-occurring combinations of resource properties as *recurring resource styles*.

Fourney and Terry emphasised that tutorial content must be formalized, because varying communication techniques pose a challenge to machine understanding and generation of tutorials [75]. Mehlenbacher elicited that documentation development involves establishing *design goals* to ensure usability of documentation [34]. In prior work, we found that resource seekers have *preferences* about the style of resources they would like to access [13], and use *cues* to find such resources. To support resource creation and resource seeking, we propose a framework for systematically reasoning about the design of software tutorial resources.

VII. Conclusion

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 reveal 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 formalize 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 leverage our framework to implement three techniques to identify relevant resource styles. We apply 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.

This works contributes an understanding of the current state of software technology tutorials, a framework for characterizing resources and the resource landscape based on design properties, and practical applications of this framework. Resource creators can use our framework to make decisions about whether to align with or deviate from the styles of existing resources. Resource seekers can use the resource styles to elicit their requirements for a learning resource. Our findings can also inform an augmented search system wherein prominent resource styles can be displayed on a search results page as informative potential *cues* for programmers to use during their search. Our framework can be modified to investigate different aspects of the software tutorial landscape in a context-specific manner. For example, we can leverage the recurring styles to determine whether certain styles are frequent among resources that cover particular technology topics. Additionally, our framework for reasoning about the design of resources can be applied to other types of resources for other technologies.

ACKNOWLEDGEMENTS

We thank Mathieu Nassif, Jessie Galasso-Carbonnel, and the anonymous reviewers for their valuable insights and feedback. This work is funded by the Natural Sciences and Engineering Research Council of Canada (NSERC).

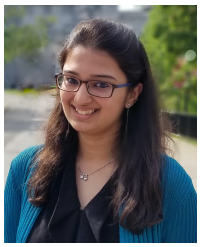
RESOURCE REFERENCES

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://beginnersbook.com/2013/03/constructors-in-java/
5. https://www.javatpoint.com/convert-object-to-array-in-javascript
6. https://www.w3schools.blog/union-type-typescript
7. https://www.javatpoint.com/instance-initializer-block
8. https://www.javatpoint.com/how-to-enable-javascript-in-my-browser
9. https://www.javatpoint.com/design-patterns-c-sharp
10. https://www.guru99.com/date-time-and-datetime-classes-in-python.html
11. https://www.tutorialspoint.com/javascript/javascript_events.htm
12. http://csharp.net-informations.com/gui/cs-scrollbars.htm
13. https://www.educba.com/logical-operators-in-c-sharp/
14. https://docs.oracle.com/javase/tutorial/collections/interfaces/order.html
15. https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures
16. https://www.tutorialkart.com/typescript/typescript-switch-
17. https://www.tutorialspoint.com/python/python_sending_email.htm
18. https://www.guru99.com/java-platform.html
19. https://www.javatpoint.com/how-to-install-python
20. http://csharp.net-informations.com/statements/enum.htm
21. https://www.tutorialspoint.com/java/java_quick_guide.htm
22. https://www.w3schools.blog/class-in-typescript
23. https://www.typescripttutorial.net/typescript-tutorial/typescript-class-
24. https://www.geeksforgeeks.org/switch-statement-in-c-sharp-
25. https://www.tutorialsteacher.com/csharp/csharp-switch
26. https://docs.python.org/3/tutorial/appendix.html
27. https://docs.oracle.com/javase/tutorial/essential/regex/quant.html
28. https://docs.oracle.com/javase/tutorial/collections/interfaces/order.html
29. https://www.tutorialspoint.com/javascript/javascript_operators.htm

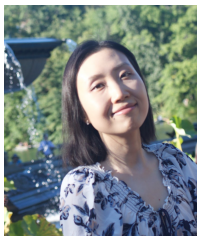
REFERENCES

[1] C. Parnin and C. Treude, "Measuring API documentation on the web," in *Proceedings of the International Workshop on Web 2.0 for Software Engineering*, 2011.
[2] V. Phoha, "A standard for software documentation," in *Computer*, 1997.
[3] R. Ries, "IEEE standard for software user documentation," in *Proceedings of the International Conference on Professional Communication, Communication Across the Sea: North American and European Practices*, 1990.
[4] "Ieee standard for information technology–systems design–software design descriptions," *IEEE STD 1016-2009*, 2009.
[5] H. Van Der Meij and M. Gellevij, "The four components of a procedure," *IEEE Transactions on Professional Communication*, 2004.
[6] D. Procida, "Diátaxis documentation framework." [Online]. Available: https://diataxis.fr/
[7] B. Dagenais and M. P. Robillard, "Creating and evolving developer documentation: Understanding the decisions of open source contributors," in *Proceedings of the International Symposium on Foundations of Software Engineering*, 2010.
[8] D. M. Arya, M. Nassif, and M. P. Robillard, "A data-centric study of software tutorial design," *IEEE Software*, 2021.
[9] R. Tiarks and W. Maalej, "How does a typical tutorial for mobile development look like?" in *Proceedings of the Working Conference on Mining Software Repositories*, 2014.
[10] P. Pirolli and S. Card, "Information foraging," *Psychological Review*, 1999.
[11] H. Li, Z. Xing, X. Peng, and W. Zhao, "What help do developers seek, when and how?" in *Proceedings of the Working Conference on Reverse Engineering*, 2013.
[12] J. Escobar-Avila, D. Venuti, M. Di Penta, and S. Haiduc, "A survey on online learning preferences for computer science and programming," in *Proceedings of International Conference on Software Engineering: Software Engineering Education and Training*, 2019.
[13] D. M. Arya, J. L. C. Guo, and M. P. Robillard, "How programmers find online learning resources," *Empirical Software Engineering*, 2022.
[14] M. Käki, "Findex: Search result categories help users when document ranking fails," *Technology, Safety, Community: Conference on Human Factors in Computing Systems*, 2005.
[15] M. Käki and A. Aula, "Findex: Improving search result use through automatic filtering categories," *Interacting with Computers*, 2005.
[16] X. Jin, N. Niu, and M. Wagner, "Facilitating end-user developers by estimating time cost of foraging a webpage," in *IEEE Symposium on Visual Languages and Human-Centric Computing*, 2017.
[17] H. van der Meij and J. van der Meij, "A comparison of paper-based and video tutorials for software learning," *Computers & Education*, 2014.
[18] A. Hora, "Googling for software development: What developers search for and what they find," in *Proceedings of the International Conference on Mining Software Repositories*, 2021.
[19] R. Ward, "The content and organisation of user documentation for information systems," in *Colloquium on Issues in Computer Support for Documentation and Manuals*, 1993.
[20] J. P. Kincaid, R. P. Fishburne, R. L. Rogers, and B. S. Chissom, "Derivation of new readability formulas (automated readability index, fog count and flesch reading ease formula) for navy enlisted personnel," 1975.
[21] R. P. Buse and W. R. Weimer, "Learning a metric for code readability," *IEEE Transactions on Software Engineering*, 2010.
[22] J. Karreman, N. Ummelen, and M. Steehouder, "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*, 2005.
[23] M. Nassif and M. P. Robillard, "Wikifying software artifacts," *Empirical Software Engineering*, 2021.
[24] C. Treude, M. P. Robillard, and B. Dagenais, "Extracting development tasks to navigate software documentation," *IEEE Transactions on Software Engineering*, 2015.
[25] W. Maalej and M. P. Robillard, "Patterns of knowledge in API reference documentation," *IEEE Transactions on Software Engineering*, 2013.
[26] M. P. Robillard and R. Deline, "A field study of API learning obstacles," *Empirical Software Engineering*, 2011.
[27] A. T. Ying and M. P. Robillard, "Selection and presentation practices for code example summarization," in *Proceedings of the Foundations of Software Engineering*, 2014.
[28] H. Abdi, "Bonferroni and Šidák corrections for multiple comparisons," *Encyclopedia of measurement and statistics*, 2007.
[29] E. R. Girden, *ANOVA: Repeated measures*. Sage, 1992.
[30] P. Rani, S. Abukar, N. Stulova, A. Bergel, and O. Nierstrasz, "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*, 2021.
[31] S. Inzunza, R. Juárez-Ramírez, and S. Jiménez, "API documentation: A conceptual evaluation model," in *Advances in Intelligent Systems and Computing*, 2018.
[32] M. Meng, S. Steinhardt, and A. Schubert, "Application programming interface documentation: What do software developers want?" *Journal of Technical Writing and Communication*.
[33] P. Pirolli and W.-t. Fu, "SNIF-ACT: A model of information foraging on the world wide web," in *Proceedings of the International Conference on User Modeling*, 2003.
[34] B. Mehlenbacher, "Documentation: not yet implemented, but coming soon," in *The HCI Handbook: Fundamentals, Evolving Technologies, and Emerging Applications*, 2003.
[35] M. P. Robillard, "What makes APIs hard to learn? the answers of developers," *Software, IEEE*, 2009.
[36] E. Aghajani, C. Nagy, O. L. Vega-Márquez, M. Linares-Vásquez, L. Moreno, G. Bavota, and M. Lanza, "Software documentation issues unveiled," in *Proceedings of the International Conference on Software Engineering*, 2019.
[37] U. Priss, "Formal concept analysis in information science," *Annual Review of Information Science and Technology*, 2006.
[38] D. I. Ignatov, *Introduction to formal concept analysis and its applications in information retrieval and related fields*, ser. Communications in Computer and Information Science, Cham, 2015.
[39] J. Li, K. Cheng, S. Wang, F. Morstatter, R. P. Trevino, J. Tang, and H. Liu, "Feature selection: A data perspective," *ACM Computing Surveys*, 2017.
[40] J. M. Cigarrán, J. Gonzalo, A. Peñas, and F. Verdejo, "Browsing search results via formal concept analysis: Automatic selection of attributes," in *Concept Lattices*, 2004.
[41] J. Cigarran, A. Castellanos, and A. Garcia-Serrano, "A step forward for

topic detection in twitter: An fca-based approach," *Expert Systems with Applications*, 2016.
[42] T. Hanika, M. Koyda, and G. Stumme, "Relevant attributes in formal contexts," Dec 2018. [Online]. Available: https://arxiv.org/abs/1812.08868v1
[43] A. Castellanos, J. Cigarrán, and A. García-Serrano, "Formal concept analysis for topic detection: A clustering quality experimental analysis," *Information Systems*, 2017.
[44] O. Nenadic and M. Greenacre, "Correspondence analysis in r, with two- and three-dimensional graphics: The ca package." 2007.
[45] S. M. Dias and N. J. Vieira, "Concept lattices reduction: Definition, analysis and classification," *Expert Systems with Applications*, 2015.
[46] N. Jay, F. Kohler, and A. Napoli, "Analysis of social communities with iceberg and stability-based concept lattices," in *Formal Concept Analysis*, ser. Lecture Notes in Computer Science, 2008.
[47] S. Kuznetsov, "On stability of a formal concept," *Annals of Mathematics and Artificial Intelligence*, 2007.
[48] S. Kuznetsov, S. Obiedkov, and C. Roth, "Reducing the representation complexity of lattice-based taxonomies," in *Conceptual Structures: Knowledge Architectures for Smart Applications*, 2007.
[49] C. Roth, S. Obiedkov, and D. Kourie, "Towards concise representation for taxonomies of epistemic communities," in *Concept Lattices and Their Applications*, 2008.
[50] R. H. Earle, M. A. Rosso, and K. E. Alexander, "User preferences of software documentation genres," in *Proceedings of the International Conference on the Design of Communication*, 2015.
[51] M. P. Robillard, A. Marcus, C. Treude, G. Bavota, O. Chaparro, N. Ernst, M. A. Gerosa, M. Godfrey, M. Lanza, M. Linares-Vásquez, G. C. Murphy, L. Moreno, D. Shepherd, and E. Wong, "On-demand developer documentation," in *International Conference on Software Maintenance and Evolution*, 2017.
[52] S. Chen, Z. Feng, and X. Yi, "A general introduction to adjustment for multiple comparisons," *Journal of Thoracic Disease*, 2017.
[53] P. Chatterjee, M. A. Nishi, K. Damevski, V. Augustine, L. Pollock, and N. A. Kraft, "What information about code snippets is available in different software-related documents? an exploratory study," in *Proceedings of the International Conference on Software Analysis, Evolution and Reengineering*, 2017.
[54] N. Ummelen, "The selection and use of procedural declarative information in software manuals," *Journal of Technical Writing and Communication*, 1996.
[55] G. Angelini, "Current practices in web API documentation," in *European Academic Colloquium on Technical Communication*, 2018.
[56] A. Head, J. Jiang, J. Smith, M. A. Hearst, and B. Hartmann, "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*, 2020.
[57] H. Tang and S. Nadi, "Evaluating software documentation quality," in *Proceedings of the International Conference on Mining Software Repositories*, 2023.
[58] G. Ajam, C. Rodriguez, and B. Benatallah, "Scout-bot: Leveraging API community knowledge for exploration and discovery of API learning resources," *CLEI electronic journal*, 2021.
[59] M. Meng, S. M. Steinhardt, and A. Schubert, "Optimizing API documentation: Some guidelines and effects," in *Proceedings of the International Conference on Design of Communication*, 2020.
[60] B. Dagenais and M. P. Robillard, "Using traceability links to recommend adaptive changes for documentation evolution," *IEEE Transactions on Software Engineering*, 2014.
[61] D. M. Arya, J. L. C. Guo, and M. P. Robillard, "Information correspondence between types of documentation for APIs," *Empirical Software Engineering*, 2020.
[62] R. E. Johnson, "Documenting frameworks using patterns," in *Proceedings of the Conference on Object-oriented Programming Systems, Languages, and Applications*, 1992.
[63] G. Butler, R. K. Keller, and H. Mili, "A framework for framework documentation," *ACM Computing Surveys*, 2000.
[64] G. Butler, P. Grogono, and F. Khendek, "A reuse case perspective on documenting frameworks," in *Proceedings of the Asia Pacific Software Engineering Conference*, 1998.
[65] B. Ganter, R. Wille, and C. Franzke, *Formal Concept Analysis: Mathematical Foundations*. Springer-Verlag, 1997.
[66] R. Al-Msie'Deen, A. Seriai, M. Huchard, C. Urtado, S. Vauttier, and H. Eyal-Salman, "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*, 2013.
[67] R. AL-Msie'deen, A. Seriai, M. Huchard, C. Urtado, S. Vauttier, and H. E. Salman, "Feature location in a collection of software product variants using formal concept analysis," in *Safe and Secure Software Reuse*, 2013.
[68] J. Galasso-Carbonnel, M. Huchard, A. Miralles, and C. Nebut, "Feature model composition assisted by formal concept analysis," in *Proceedings of the International Conference on Evaluation of Novel Approaches to Software Engineering*, 2017.
[69] J. Galasso-Carbonnel, M. Huchard, and C. Nebut, "Analyzing variability in product families through canonical feature diagrams," in *Proceedings of the International Conference on Software Engineering and Knowledge Engineering*, 2017.
[70] R. Godin and H. Mili, "Building and maintaining analysis-level class hierarchies using galois lattices," in *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications*, 1993.
[71] F. Buchli, "Detecting software patterns using formal concept analysis," Master's thesis, 2003.
[72] T. Tilley, R. Cole, P. Becker, and P. Eklund, "A survey of formal concept analysis support for software engineering activities," in *Formal Concept Analysis: Foundations and Applications*, 2005.
[73] S. Ferré, M. Huchard, M. Kaytoue, S. O. Kuznetsov, and A. Napoli, "Formal concept analysis: From knowledge discovery to knowledge processing," in *A Guided Tour of Artificial Intelligence Research: Volume II: AI Algorithms*, 2020.
[74] P. Valtchev, R. Missaoui, and R. Godin, "Formal concept analysis for knowledge discovery and data mining: The new challenges," in *Concept Lattices*, 2004.
[75] A. Fourney and M. Terry, "Mining online software tutorials: Challenges and open problems," in *Proceedings of the Conference on Human Factors in Computing Systems*, 2014.



Deeksha M. Arya is a Ph.D. student in Computer Science at McGill University. She works on investigating how the search process of programmers can be supported as they navigate the complex software documentation landscape to find resources pertinent to their needs.



Jin L.C. Guo is an Assistant Professor of Computer Science at McGill University. She is interested in the intersection between Software Engineering, Human-Computer Interaction, and Artificial Intelligence. Her recent projects in particular focus on software traceability, OSS usability, and software documentation.

Martin P. Robillard is a Professor in the School of Computer Science at McGill University. He received his Ph.D. in Computer Science from the University of British Columbia. His research is in the area of software engineering, with an emphasis on the human-centric aspects of software development.