Noname manuscript No.
(will be inserted by the editor)

Information Correspondence between Types of Documentation for APIs

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

Received: date / Accepted: date

Abstract Documentation for programming languages and their APIs takes many forms, such as reference documentation, blog posts or other textual and visual media. Prior research has suggested that developers switch between reference and tutorial-like documentation while learning a new API. Documentation creation and maintenance is also an effort-intensive process that requires its creators to carefully inspect and organize information, while ensuring consistency across different sources. This article reports on the relationship between information in tutorials and in API reference documentation of three libraries on the topics: regular expressions, unified resource location and Input/Output in the two programming languages Java and Python. Our investigation reveals that about half of the sentences in the tutorials studied describe *API Information*, i.e. syntax, behaviour, usage and performance of the API, that could be found in the reference documentation. The remaining are tutorial specific use-cases and examples. We also elicited and analyzed six types of correspondences between sentences in tutorials and reference documentation, ranging from *identical* to *implied*. Based on our findings, we propose a general information reuse pattern as a structured abstraction to represent the systematic integration of information from the reference documentation into a tutorial. We report on the distribution of 38 instances of this pattern, and on the impact of applying the pattern automatically on the existing tutorials. This work lays a foundation for understanding the nature of information

Deeksha Arya
McGill University
E-mail: deeksha.arya@mail.mcgill.ca

Jin L.C. Guo
McGill University
E-mail: jguo@cs.mcgill.ca

Martin P. Robillard
McGill University
E-mail: martin@cs.mcgill.ca

correspondence across different documentation types to inform and assist documentation generation and maintenance.

Keywords Software Documentation · Application Programming Interface · Qualitative Analysis · Exploratory Study

1 Introduction

Complete, correct and consistent documentation for application programming interfaces (APIs) is a boon for developers to understand the functionalities of software and to master their usage. API documentation may be presented in many forms, including reference documentation and tutorials. We define *reference documentation* as a dictionary-style document indexed by API element and intended mostly for retrieving specific technical information. In contrast, we define a *tutorial* as a document intended primarily to teach its users how to use a technology, without specific constraints on the structure of the document (such as a direct mapping to the underlying API).

Despite the availability of documentation, Robillard (2009) and Uddin and Robillard (2015) observed that API reference documentation lacks, in many cases, the kind of information that users seek, such as examples and how-to-use descriptions. Meng et al. (2018, 2019) discovered that developers often have to switch between reference and tutorial documentation during software development to retrieve the information they need. There is thus evidence that developers need different types of documentation when learning and using technology. To fulfill this need, popular technology platforms are usually supported by a documentation ecosystem that includes a variety of document types that synergistically complement each other. Although it is not rare to observe extensive information overlap between document types, we are not aware of any documentation creation or maintenance process where the correspondence between different documentation types is taken into account. More generally, very little is known about the correspondence between different types of API documentation.

To improve our understanding of documentation ecosystems and inform the future development of advanced documentation systems, we conducted a case study to explore the question: *what is the correspondence between information contained in tutorials and reference documentation for application programming interfaces (APIs)?* Based on an initial observation that the structure and style of tutorial documentation is not uniform even across tutorials by a single technology provider, we structured our inquiry as a multiple case study (Runeson et al., 2012), investigating the relationship between the API reference documentation and tutorials for three API-supported features (regular expressions, URL handling, and Input/Ouput) in two programming languages (Java and Python). For analyzing each case, we relied on a systematic manual analysis of the content of tutorials as reference documentation, following a qualitative content analysis (Krippendorff, 2018).

We followed an inductive research process organized in three phases. First, we determined the API library topics and identified the corresponding tutorial and reference documentation (Section 2). Second, for each sentence in the six tutorials, we manually analyzed whether it constituted *API Information* or *Supporting Text* (see Section 3.1). If the sentence constituted *API Information*, we determined whether there was any sentence in the reference documentation that contained the same information as the tutorial sentence. We classified each identified sentence pair from tutorials and reference documentation according to the extent to which the two sentences match (Section 3.2). Finally, we identified frequently occurring structures in the tutorials that map to information in reference documentation as instances of an information reuse pattern between tutorials and reference documentation (Section 4).

Our detailed analysis of 2 445 tutorial sentences revealed that, depending on the tutorial, between 46% and 76% of the tutorial sentences constitute *API Information*. The nature of the correspondence between these sentences and those in the reference documentation varies across a spectrum from *identical* to *implied*, with between 15% to 45% of *API Information* tutorial sentences remaining *unmatched*. We devised an abstract structure that systematical integrates API reference information into tutorials. We call this structure an *information reuse pattern*. We noted 38 instances of this pattern across four major API element types in the six tutorials studied, indicating a practice of *systematic* reuse of information across document types. Further more, we discovered for each tutorial, over 64% of the individual sentences not covered by a pattern instance are found at *arbitrary* or low-prominence locations in the reference documentation.

This study is the first to report on a systematic analysis of the correspondence between tutorials and reference documentation for APIs that employs a painstaking manual classification process to take into account the nuances of differences in sentence semantics. This imperative process cannot be replaced by existing natural language processing techniques, which are incapable of accurately determining the level of relatedness between highly technical text (Cleland-Huang and Guo, 2014; Al Omran and Treude, 2017). Our findings provide empirically-based insights about the content of popular tutorials and their correspondences in reference documentation. In particular, we document how the information in documentation is currently related, and provide insights on how the level of correspondence can be further increased and supported by more advanced technology, promoting consistency across documentation types as a favorable by-product.

Data artifact

This article is complemented by a data artifact available on-line at `https://doi.org/10.5281/zenodo.3959240`. This artifact comprises data annotation guidelines, and our data set.

Table 1: Documentation data set along with their associated topic, corresponding resources and the number of sentences we extracted from the tutorials. For reference documentation, the resources include all API elements under the stated package or module. For tutorials, we state the title of the tutorial.

Language	Doc Type	Topic	Resource	# of Sentences
Java Platform SE v8	Reference	REGEX	*java.util.regex* [3]	-
		URL	*java.net* [4]	-
		I/O	*java.nio.file* [5]	-
	Tutorial	REGEX	*Lesson: Regular Expressions* [6]	402
		URL	*Lesson: Working with URLs* [7]	183
		I/O	*File I/O (Featuring NIO.2)* [8]	1200
Python v3.7.2	Reference	REGEX	*re* [9]	-
		URL	*urllib* [10]	-
		I/O	*Built-in Functions* [11]	-
	Tutorial	REGEX	*Regular Expression HOWTO* [12]	402
		URL	*HOWTO Fetch Internet Resources Using The urllib Package* [13]	135
		I/O	*Input and Output* [14]	123

2 Data Collection

We selected our cases across two dimensions - *programming languages and their APIs* and *topic* (or functionality supported by the API). For programming languages, we selected Java and Python because they are two popular languages that support general application development yet have significant differences in syntax and library organization. For topics, we selected three basic functionalities likely to be used in a large variety of development contexts: Regular Expressions (`REGEX`), URL connectivity and handling (`URL`) and Input/Output (`I/O`). This selection introduces diversity in our cases in terms of writing style, information content and length (see Table 1). Henceforth, we use the format of programming language–topic to refer any of our six cases. For example, Java–`REGEX` refers to the tutorial and reference documentation of regular expressions API in Java, while `*`–`I/O` refers to the tutorial and reference documentation of input and output API across both programming languages.

For each topic, we retrieved the corresponding reference and tutorial documentation resources (HTML files) for the Java Platform Standard Edition version 8 and Python version 3.7.2. The organization of the functional topics in the documentation resources is different between Java and Python. For example, connection via sockets is provided within the `java.net` package [1]† whereas, in Python, `socket` [2] is a separate module, outside the `urllib` package.

Among our target documentation resources, we transformed every tutorial file into a sequence of sentences. Splitting sentences from HTML documents is not a trivial problem. To preserve the accuracy of sentence splitting, we

†Links to documentation are referenced at the end of this article under "References to Documentation Sources".

designed heuristics to handle documentation files for Java and Python and proceeded to manually verify the results. We describe the heuristics in Appendix A. We also replaced code blocks by a single token CODE for sentence readability, but did not alter inline code. Finally, because the quality of the sentence data set is of high importance in our study, we manually corrected all incorrect splits, which involved merging seven and splitting 76 data items produced by the splitting algorithm. Table 1 lists the documentation elements we studied, along with their associated topic, corresponding resource files and the final number of sentences we extracted from the tutorials. Our study design did not require that we split sentences in the reference documentation (see Section 3).

3 Information Correspondence between Documentation Types

We analyzed to what extent information contained in tutorials corresponds to information present in the reference documentation. We used the *tutorial sentence* as a unit of information which we match to reference documentation. Although, in some cases, reference documentation is created first and other documentation types are produced as the project evolves (Dagenais and Robillard, 2010, p. 6), this part of our analysis (the correspondence between documentation types) is not dependent on the temporal order in which documents of different types are created.

3.1 Initial Classification of Tutorial Sentences

Not all sentences in a tutorial capture information about an API. A certain fraction of the text is employed for general pedagogical purposes, way-pointing, segues, and to dissect specific example scaffolding. For instance the following sentence provides general background and motivation for some of the tutorial's content: "You must learn a specific syntax to create regular expressions—one that goes beyond the normal syntax of the Java programming language." [15] For sentences that do not capture information about the API, it is unlikely that we would find a corresponding sentence in the reference documentation, and even then such a correspondence would be spurious. To determine a baseline for the information that *can* be matched between document types, we manually classified each sentence in the tutorials as either capturing *API Information* or containing *Supporting Text*. For a tutorial sentence to qualify as *API Information*, it must describe the syntax, behaviour, usage, performance and/or support of the API under consideration. For example, we categorize the following sentence as *API Information* because it documents the behavior of the matching functions for regular expressions: "By default, case-insensitive matching assumes that only characters in the US-ASCII charset are being matched." [16]

We designed a coding guide to discriminate sentences between *API Information* and *Supporting Text* by following an iterative content analysis process

Table 2: Frequency of sentences containing API Information in each tutorial. The last row and column each contain the average percentage and in parentheses, the total number of tutorial sentences containing the relation type across the respective row and column.

Language	Relation to API	REGEX	URL	I/O	Average (Total)
Java	API Information	46% (183)	58% (107)	52% (619)	51% (909)
	Supporting Text	54% (219)	42% (76)	48% (581)	49% (876)
Python	API Information	49% (198)	76% (102)	62% (76)	57% (376)
	Supporting Text	51% (204)	24% (33)	38% (47)	43% (284)
Average (Total)	API Information	47% (381)	66% (209)	53% (695)	53% (1285)
	Supporting Text	53% (423)	34% (109)	47% (628)	47% (1160)

Table 3: Contingency Matrix of annotations of a random, stratified sample of 332 tutorial sentences between the author and the external annotator.

		External Annotator	
		API Information	*Supporting Text*
Author	*API Information*	121	52
	Supporting Text	20	139

whereby three authors first independently coded a subset of 195 sentences from *–REGEX tutorials and resolved issues and disagreements. After improving the coding guide, the first author coded the remainder of the sentences as either *API Information* or *Supporting Text*. Table 2 shows the frequency of sentences containing API Information in each tutorial.

The annotation process we followed is not devoid of subjectivity. We measured the subjectivity of the task by asking an annotator not associated with this project to independently code a stratified random sample of 332 sentences.[‡] Our observed agreement with the independent annotator was $260/332 = 0.783$, yielding a Cohen's kappa value of 0.57, which quantifies the subjectivity of the task. Table 3 shows the direction of the disagreements: our analysis was more liberal in including sentences as API information. Sentences marked as *API Information* by the first author but not by the external annotator frequently described information about an underlying concept or its use-case such as "An absolute URL contains all of the information necessary to reach the resource in question."[17]. Section 3.4 discusses the implications of the subjectivity of this initial classification task.

[‡]We selected a sample size of 332 using as guide the sample size for a minimum confidence interval of 5% for the proportion of sentences in agreement based on simple random sampling. We used stratified sampling, where a number of sentences is randomly drawn from each of the six tutorials in proportion to the number of sentences in the tutorial. This prevents the calculation of an exact confidence interval.

For each tutorial sentence (S):

1. If S contains a hyperlink to the reference documentation, we follow the hyperlink and proceed to Step 4.
2. We search for a hyperlink in relevant neighboring sentences of S; if we find one, we follow it and proceed to Step 4.
3. We identify elements such as class, method or constant names in S or its closest relevant neighbors. We then refer to the reference documentation containing the elements identified and proceed to Step 4.
4. We inspect the reference documentation for S. If a match is found, we proceed to Step 7.
5. We inspect the reference documentation for coherent subphrases of S until a match to it is found or all meaningful subphrases have been exhausted. If a match is found, we proceed to Step 7.
6. We search for a sentence that provides the same information as S in the reference documentation. If found, we proceed to Step 7. Otherwise, we proceed to Step 10.
7. For each matched sentence, we assess if it provides the same information as S. If it does not, we discard the reference documentation sentence.
8. If no reference documentation sentences remain, we proceed to Step 10.
9. We identify the sentence among the matched sentences closest in correspondence to S. We discard all other reference documentation sentences.
10. We assess the extent to which the matched sentence corresponds to S, or annotate the tutorial sentence as *Unmatched*.

Fig. 1: Procedure used to find correspondences of tutorial *API Information* sentences in the relevant reference documentation.

3.2 Matching Tutorial Sentences to Reference Documentation

For each of the 1285 tutorial sentences we labeled as capturing API information, we attempted to match it to a corresponding sentence in the reference documentation.

We chose to manually match the sentences for two main reasons. First, an extensive knowledge of the target API is required to fully understand the content of the sentences and to establish their relations. For example, the sentence in the Python–REGEX tutorial: "Groups are numbered starting with 0."[12] refers to the behaviour of the method `group()`. In contrast, a misleadingly similar sentence in the reference documentation refers to the concept of groups within regular expression syntax: "Groups are numbered starting from 1"[9]. Second, the extent to which two sentences in different documents can relate to each other varies greatly.[§] We thus devised the matching process described in Fig 1.

The first author followed this process to examine all the tutorial sentences containing API information. As a result, we identified the following seven types

[§] We experimented with an automated technique based on computing the Jaccard index between sentences. Not surprisingly, the precision and recall were so low that producing a reliable analysis required a manual review as onerous as a complete manual inspection.

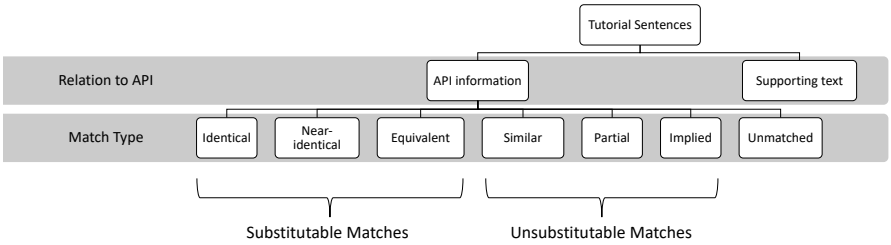


Fig. 2: Taxonomy for tutorial sentences

of correspondences between *API Information* sentences in tutorials and their corresponding reference documentation. Fig 2 summarizes our taxonomy for tutorial sentences.

Identical: The tutorial sentence is a verbatim copy of a sentence in the reference documentation. For example, in Java–REGEX:

Tutorial:	**Reference documentation:**
By default, matching does not take canonical equivalence into account. [16]	By default, matching does not take canonical equivalence into account. [18]

Near-identical: The tutorial sentence is exactly the same as a sentence in the reference documentation except for minor variations in choice of words that do not alter the meaning of the sentence. The difference between the following sentences from Python–REGEX are in italics:

Tutorial:	**Reference documentation:**
Usually ^ matches only at the beginning of the string, and $ matches only at the end of the string and immediately before the newline (if any) at the end of the string. [12]	*By default*, '^' matches only at the beginning of the string, and '$' only at the end of the string and immediately before the newline (if any) at the end of the string. [9]

Equivalent: The tutorial sentence is semantically equivalent with one in the reference documentation, but is phrased in a different way, e.g., it uses the passive voice instead of the active. Equivalent sentences can be substituted with each other with no impact on the coherence of the host text. In Python–URL, the following sentences describe the default behaviour of the `Request` method in `urllib.request`:

Tutorial:	**Reference documentation:**
If you do not pass the data argument, urllib uses a GET request. [13]	The default is 'GET' if data is None or 'POST' otherwise. [10]

Similar: Although the tutorial sentence is semantically equivalent with one in the reference documentation, the two sentences cannot be substituted for each other because it would impact the coherence of the text or add excessive information to the tutorial. Additional details on the classification criterion can be found in Appendix B. The following Java–URL tutorial sentence describes a

particular example before providing information about the API (in italics). If replaced by the API reference documentation sentence, the reference to the example that prompts this information would be lost. However, the non-italicised sentence fragment contains *Supporting Text*, and so this is not considered a *partial* correspondence.

Tutorial:	**Reference documentation:**
Encoding the special character(s) in this example is easy as there is only one character that needs encoding, but for URL addresses that have several of these characters or if you are unsure when writing your code what URL addresses you will need to access, *you can use the multi-argument constructors of the java.net.URI class to automatically take care of the encoding for you.* [17]	The multi-argument constructors quote illegal characters as required by the components in which they appear. [19]

Partial: Some, but not all, of the information in the tutorial sentence is present in the reference documentation. The Java–REGEX documentation contains the sentence below. The reference documentation explicitly states that the `lookingAt()` method begins at the beginning of the input string, but it does not describe anywhere that the `matches()` method does too. Hence only some of the information provided in the tutorial sentence can be sourced from the reference documentation.

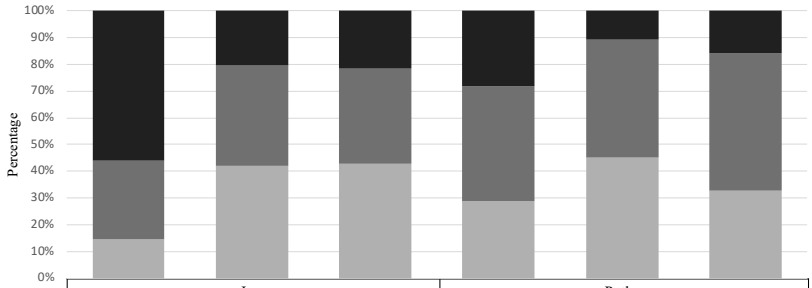
Tutorial:	**Reference documentation:**
Both methods [lookingAt() and matches()] always start at the beginning of the input string. [20]	public boolean lookingAt() Attempts to match the input sequence, starting at the beginning of the region, against the pattern. [21]

Implied: The information in a tutorial sentence can be implied from information in the reference documentation, provided the reader has the relevant domain knowledge. Identifying this type of correspondence can be subjective as it depends on the experience and expertise of a reader. The sentence pair below is from Java–IO. Here, the tutorial states that the `varargs` argument of the `Files.readAttributes` methods accept the constant `NOFOLLOW_LINKS`. The API reference documentation makes no explicit statement, but describes the method's behaviour if this constant was to be passed as an argument. An experienced user would likely be able to infer that `NOFOLLOW_LINKS` is an acceptable input and should be passed in the `varargs` argument.

Tutorial:	**Reference documentation:**
The varargs argument currently supports the LinkOption enum, NOFOLLOW_LINKS. [22]	If the option NOFOLLOW_LINKS is present then symbolic links are not followed. [23]

Unmatched: Sentences we classified as capturing *API Information* in Section 3.1, but were unable to locate any corresponding sentences in the reference documentation.

Fig 3 shows the frequency of different types of correspondence relative to all the sentences capturing *API information*. In Java–REGEX, 37% of these sentences are *identical* to sentences in the reference documentation. In contrast,



	Java			Python		
	REGEX	URL	I/O	REGEX	URL	I/O
Identical	67 (37%)	0 (0%)	1 (0%)	14 (7%)	1 (1%)	0 (0%)
Near-identical	15 (8%)	2 (2%)	27 (4%)	10 (5%)	0 (0%)	0 (0%)
Equivalent	20 (11%)	20 (19%)	105 (17%)	32 (16%)	10 (10%)	12 (16%)
Substitutable	56%	21%	21%	28%	11%	16%
Similar	19 (10%)	23 (21%)	138 (22%)	43 (22%)	23 (22%)	28 (37%)
Partial	1 (0%)	1 (1%)	16 (3%)	6 (3%)	5 (5%)	2 (2%)
Implied	34 (19%)	16 (15%)	65 (11%)	36 (18%)	17 (17%)	9 (12%)
Unsubstitutable	29%	37%	36%	43%	44%	51%
Unmatched	27 (15%)	45 (42%)	267 (43%)	57 (29%)	46 (45%)	25 (33%)

Fig. 3: Frequency of correspondence between *API Information* sentences in tutorials and sentences in reference documentation.

in Python–REGEX, only 7% of the sentences are *identical* matches. *–URL tutorials paint a different picture. We did not find any *identical* correspondences in Java–URL, and only one in Python–URL. Our observations in *–I/O documentation are similar to *–URL. In Java, only one *identical* correspondence exists whereas Python has none. The percentage of *unmatched* sentences in Python–IO lies in-between that in Python–REGEX and Python–URL with 33% cases but form the majority in Java with 43% occurrences. *Equivalent* and *similar* matches occur nearly equally frequently in Java. In all six cases, *near-identical* and *partial* matches are infrequent (at most 8% and 5% respectively).

Our taxonomy characterizes the correspondence between matched pairs into a spectrum from *identical* to *implied*. We further split this spectrum into two coarse-grained categories: *substitutable* and *unsubstitutable* correspondences (see Fig 2).

- *Substitutable*: refers to a pair of sentences that provide the same information such that the sentence in the tutorial can be replaced by its corresponding one in the reference documentation without altering the meaning of the text or disrupting its coherence. This category includes *identical*, *near-identical* and *equivalent* matches.
- *Unsubstitutable*: refers to matched pairs of sentences where the tutorial sentence provides slightly less or more than its corresponding reference documentation sentence or can not be replaced by it in the tutorial because it would break the coherence of the text. This category includes *similar*, *partial* and *implied* matches.

The correspondence between *API Information* sentences in tutorials and reference documentation varies across programming language and topic. We found that information reuse is topic-dependent, especially in the case of *–URL, which shows similarity in the distribution of correspondence types in Java and Python. *Substitutable* matches form the majority in Java–REGEX, and correspond to a quarter of the other Java tutorials and Python–REGEX. In general, Python shows less correspondences of API documentation than Java.

Classification of Unmatched Tutorial Sentences. The proportion of unmatched tutorial sentences varies between 15% to 45% across our data set. We categorized each unmatched sentence following an open coding process, eliciting ten categories of unmatched sentences. Table 4 provides the frequency distributions, and the categories are explained in detail in Appendix C. In three tutorials, unmatched sentences predominantly describe *usage* of an API, as expected. In the other tutorials, the majority of unmatched sentences describe the *underlying topic*, *behaviour* or *internal behavior* of the API.

Table 4: Percentage distribution of categories of *unmatched* tutorial sentences rounded to the nearest integer. For each tutorial, the most frequent category is in bold. The decimals indicate exact half values and are not rounded to avoid totaling inconsistencies.

Theme	**Java**			**Python**		
	REGEX	URL	I/O	REGEX	URL	I/O
Underlying Topic	**69**	29	19	23	22	12
Behaviour	0	11	9	2	26	**52**
Usage	15	**36**	**25**	12.5	**39**	24
Internal Behavior	0	2	16	**30**	4.5	0
Use-case	8	22	13	18	4.5	8
Performance	4	0	3	12.5	2	4
Version Info	4	0	4	2	2	0
Environment	0	0	7	0	0	0
Input Configuration Details	0	0	3	0	0	0
API support	0	0	1	0	0	0

3.3 Positional Distribution of Matched Sentences in Tutorials

In addition to the degree of correspondences from tutorial sentences to sentences in the reference documentation, we also investigated the positional distribution of matched sentences within the tutorials.

Fig 4 shows a visualization of the sentence category distribution in tutorials. Each shade-coded vertical bar corresponds to the sentence classification for the sentence at the corresponding normalized position in the tutorial. Sentences are ordered from left to right.

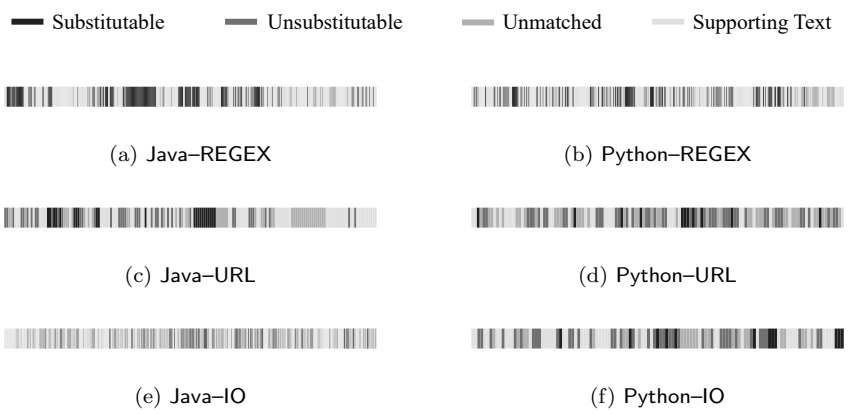


Fig. 4: Visualization of the sentence category distribution in tutorials. Each shaded vertical bar corresponds to the sentence classification for the sentence at the corresponding normalized position in the tutorial. Sentences are ordered from left to right.

At this macroscopic level, no obvious pattern is apparent, but the figure shows evidence of different styles of correspondence in the tutorials. For example, many *substitutable* correspondences in Java–REGEX take the form of large blocks of text, something we do not observe in any one of our other cases except for one instance in Java–URL. The other feature apparent from this visualization is that sentence correspondence for Python–REGEX and Java–IO is fine-grained, with most interleavings occurring at the level of individual sentences.

We illustrate how corresponding information in different document types can relate with an excerpt from the Python–REGEX case. Fig 5 shows the description of the method `sub`. The left column contains text about the `Pattern.sub` method in the tutorial [12] and the right contains a snippet from the API reference documentation [9] describing `re.sub`.

These two methods are comparable as the method definition for `Pattern.sub` in the reference documentation states that it is: "Identical to the sub() function, using the compiled pattern." [9] where `sub()` is a hyperlink reference to the documentation of `re.sub`. We can observe that the two arguments of the method have been described in a different order than in the original reference. However, the change in order of description of the arguments does not seem to hold significance in the tutorial. Further, the reference documentation contains all paragraphs (with examples) indented under the method definition. However, the same information provided in the tutorial, has indented only the first and second paragraph, while the rest align under the parent section (not shown here). Python–REGEX contains two such method description embeddings, both of which display this indentation inconsistency. Whether this is intended is difficult to assess. This observation is unique to Python–REGEX, however, as no

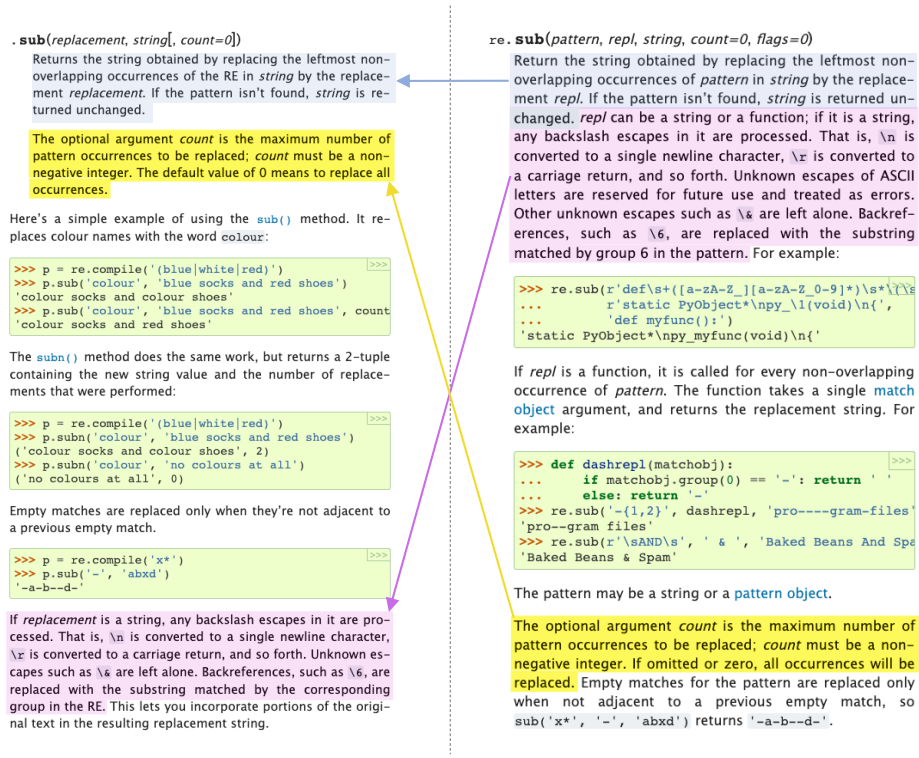


Fig. 5: Visualization of reordering of documentation during reuse across documentation types. Sentences from the reference documentation [9] (right) are reused in the tutorial [12] (left) but are reordered and inconsistently indented.

other tutorial contains such method descriptions explicitly embedded in the content.

3.4 Discussion

Our analysis reveals that between half and three quarters of the sentences in the tutorials we studied contain *API Information* (see Table 2), with the remainder consisting of *Supporting Text*. The high proportion of *API Information* of some tutorials (e.g., Python–URL, 76% of *API Information*) suggests that, in addition to *Supporting Text*, tutorials also serve the function of providing a different perspective on the information available in reference format. This observation is consistent with that of Meng et al. (2018, 2019), who noted how software developers made use of different types of documentation.

We also find a surprisingly high level of close correspondence between sentences in the reference documentation and tutorials (see Fig 3). Some of the text is so similar that we estimate that, depending on the tutorial, between

11% and 56% of the tutorial sentences that describe API information are directly substitutable by sentences from the reference documentation. This indicates that a significant amount of reuse can be achieved when creating API documentation. Furthermore, our estimates are evidence of only a fraction of the leverage that can be achieved because, as far as we know, the tutorials were created without any systematic support for content reuse (beside arbitrarily copying and pasting text). We observed that an additional 29% to 51% of tutorial sentences containing API information which, although not directly substitutable, had a corresponding sentence with related information. With support for systematic documentation reuse, it is conceivable that a fraction of these sentences could also be integrated from the reference documentation, further increasing the impact of reuse.

Our analysis of the positional distribution of matched sentences (see Fig 4) shows the absence of any obvious pattern of information correspondence at the level of the entire tutorial. For example, we do not find that matched sentences necessarily occur towards the end of a tutorial, or occur in large sequential blocks (except for a few exceptions in Java–REGEX and Java–URL). In Section 4, we expand our analysis of information correspondences to better understand how this systematic reuse of information content could be supported.

Threats to Validity

The main threat for the results described in this section is investigator bias in the coding of sentences as *API Information* or *Supporting Text* (Section 3.1). The threat originates from the fact that the task is subjective because some sentences are inherently difficult to classify. To mitigate this threat, we explicitly measured the subjectivity of the task with the help of an external annotator. The data we collected as part of this procedure help interpret the ratios described in Fig 2. Specifically, the ratio of sentences capturing API information can vary by an amount in the order of 15% (52/322 sentences not coded as *API Information* by the external annotator in disagreement with our coding). Although it will never be possible to determine the ratio of API information exactly due to the inherent subjectivity of the task, the resulting variability does not invalidate our main observation, namely that around half of a tutorial can consist of sentences describing API information. The impact of potential ambiguity in our initial classification stage is further mitigated by the second stage in our analysis, which seeks to precisely map corresponding sentences.

For the matching procedure (Section 3.2), the threat is of false negatives, namely that we may have missed some sentences in the reference documentation that would be linked to the tutorial. This threat is mitigated by the fact that tutorials are focused on a specific topic. Nevertheless, our observations on sentence correspondence can be assumed to be conservative, meaning that it is possible that some sentences we classified as *unmatched* may actually correspond to a sentence in the reference documentation. The impact of this

eventuality on our observation is limited, given that we focus our interest on matched sentences. As for threats to the classification itself, it is nonexistent for *identical* and minimal for *near-identical* and *equivalent* matches, but non-trivial for *unsubstitutable* matches. However, our conclusions are not strongly affected by unsubstitutable sentences, as we focus our analysis on substitutable ones.

Finally, our overall research method is that of the exploratory case study. Our observations avoid the threat of making observations overly influenced by a specific context because we studied six different cases across two languages, three topics, and two documentation types. However, we do not claim that the precise ratios we observed would generalize to other contexts, as indeed we observe important differences between our six cases.

4 Systematic Information Reuse

Having established the frequency of different types of sentence correspondence between tutorial and reference documentation (Fig 3), and the relative position of these correspondences (Fig 4), we investigated whether groups of substitutable sentences formed recognizable patterns. Here, we focus on the scenario in which a tutorial is created after reference documentation. We formalize our observations as an *information reuse pattern*. The pattern acts as a template for organizing sentences from reference documentation to convey information about particular aspects of the API in a tutorial. We use the pattern to aide the study of characteristics of systematic reuse of reference documentation in tutorials.

4.1 Pattern Elicitation

Fig 6 summarizes the procedure we adopted to elicit a common structure for information reuse. The concrete elicitation of the pattern enables its use in an automated manner. However, since we inferred the structure inductively, not all the instances of the pattern that we identified follow the formal structure strictly. We report on the discrepancies in Section 4.2.

We define the structure formally as the *information reuse pattern* and describe its elements in Fig 7. The pattern has three elements: the **intent** of information reuse, the **structure** using which the sentences from the reference document should be arranged in the tutorial, and the two **parameters** that users of the patterns can set to define the specific scope of the information use. The element_type parameter specifies the API element of interest for each pattern instance. In our case studies, we observed pattern instances that are relevant to four API element types: TYPE (classes, interfaces, etc.), METHOD, CONSTANT, and INPUT FORMAT OPTIONS.

Fig 8, from Java–REGEX, describes the list of accessor METHODs available for a `URL` object. The left column shows the tutorial [24], and the right col-

1. For each substitutable tutorial sentence:
   (a) We noted its immediate structural context, such as a list or paragraph within a section.
   (b) We looked up the corresponding sentence in the reference documentation and noted its immediate structural context (e.g., method or class description) and the relative location of the sentence within the documentation for this structure (e.g., leading sentence).
2. We grouped adjacent tutorial sentences positioned within the same structure and whose reference sentences laid in the same structural context. These groups form *instances of information reuse*.
3. We grouped information reuse instances extracted from the same type of structural context (e.g., class, method). These groups form our *information correspondence patterns*.
4. We abstracted a formal *structure* for representing the pattern.

Fig. 6: Procedure to capture *information reuse patterns* in tutorials.

Information Reuse Pattern	
Intent:	Provide a brief overview of a selection of cohesive API elements (classes, interfaces, methods, constants or input format options).
Structure:	List, where each item consists of the name of the API element followed by sentences extracted from the element's reference documentation.
Parameters:	1. element_type: the type of the API elements being described (eg. METHOD or CONSTANT). 2. element_descriptors: a list in which each entry is given by the tuple (element, extract), where element refers to the name of the API element and extract is a tuple (paragraph, sentences). Paragraphs and sentences are referenced using an ordinal index (e.g., 1 for the first paragraph or sentence), a range (e.g., [2-4] for the second to fourth paragraphs or sentences), or the universal quantifier * for all sentences or paragraphs. For example, (*,1) refers to the first sentence of each paragraph, whereas ([1,2],*) refers to all the sentences of the first two paragraphs.

Fig. 7: The *information reuse pattern* observed in the six tutorials under study.

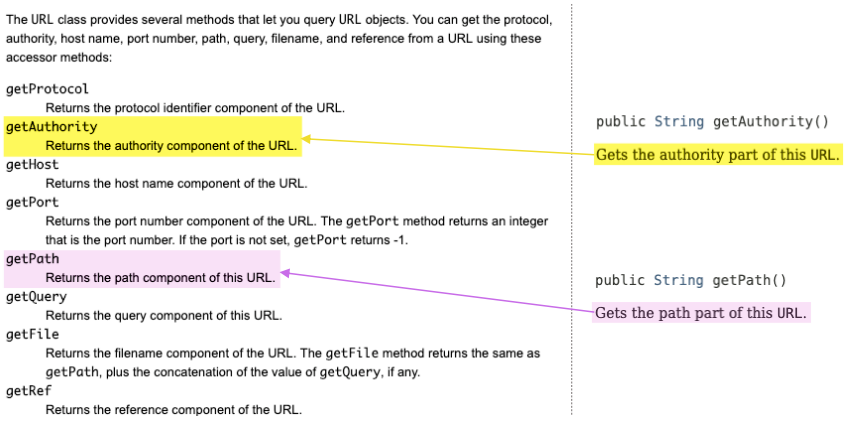


Fig. 8: An instance of information reuse for METHOD in Java–REGEX where fragments from the method description in the reference documentation [25] (right) correspond to the content in the tutorial [24] (left).

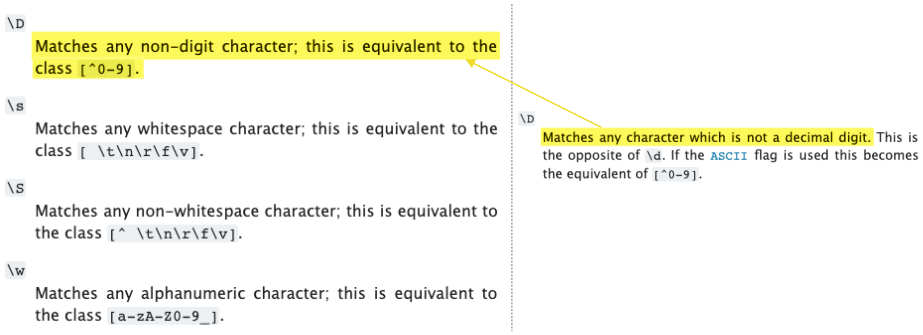


Fig. 9: Instance of information reuse for INPUT FORMAT OPTIONS in Python-REGEX where fragments from the reference documentation [12] (right) correspond to the content in the tutorial (left) [9].

umn has two fragments of the reference documentation [25] with descriptions corresponding to the selected methods.

As an example of the pattern for INPUT FORMAT OPTIONS, Fig 9 contains, on the left, the list of input format options in the Python–REGEX tutorial [12], of which the description of one option is highlighted. On the right, the source reference documentation [9] from which the sentence is reused is highlighted.

4.2 Instances of Information Reuse

Table 5 reports the distribution of pattern instances in our six cases. In the columns, "I" refers to the number of instances and "S" refers to the total number of sentences in the instances. The remaining five columns report our

Table 5: Frequency distribution of pattern instances and impact of strict pattern application. Column 'I' refers to the number of instances of the pattern, while all other columns have as values the number of sentences.

Element Type	Topic	Language	No. of		Retained	Eliminated			Added
			I	S	SC	UC	U	ST	
TYPES	REGEX	**Java**	2	4	4	0	0	0	0
		Python	0	0	-	-	-	-	-
	URL	**Java**	0	0	-	-	-	-	-
		Python	0	0	-	-	-	-	-
	I/O	**Java**	1	11	9	1	1	0	0
		Python	0	0	-	-	-	-	-
METHODS	REGEX	**Java**	8	41	41	0	0	0	11
		Python	2	7	7	0	0	0	5
	URL	**Java**	1	11	11	0	0	0	2
		Python	0	0	-	-	-	-	-
	I/O	**Java**	10	35	24	6	4	1	1
		Python	0	0	-	-	-	-	-
CONSTANTS	REGEX	**Java**	1	36	35	0	0	1	1
		Python	1	26	15	1	3	7	10
	URL	**Java**	0	0	-	-	-	-	-
		Python	0	0	-	-	-	-	-
	I/O	**Java**	5	39	22	12	5	0	7
		Python	0	0	-	-	-	-	-
INPUT FORMAT OPTIONS	REGEX	**Java**	2	6	6	0	0	0	0
		Python	3	42	14	13	3	12	-
	URL	**Java**	0	0	-	-	-	-	-
		Python	0	0	-	-	-	-	-
	I/O	**Java**	2	6	4	2	0	0	2
		Python	0	0	-	-	-	-	-

I — Number of instances,
S — Total number of sentences in all instances,
SC — Substitutable correspondences,
UC — Unsubstitutable correspondences,
U — Unmatched sentences,
ST — Supporting Text

analysis on the discrepancies between the pattern instances and the text that would be generated by applying the pattern strictly. The column *Retained - SC* contains the number of substitutable sentences that would be retained, *Eliminated - UC, U and ST* refers to the number of sentences that would be eliminated and *Added* is the number of additional API reference sentences that would be injected in the tutorial. This analysis helps us estimate the potential impact of automatically applying the information reuse pattern when creating tutorials. Eliminated sentences only comprise of unsubstitutable sentences, unmatched sentences, and *Supporting Text*. Although it is technically possible that some substitutable sentences could be lost (e.g., if they are sourced from a location outside the structure covered by a pattern), we did not observe this phenomenon in our study.

We observed that the impact of adopting our information reuse pattern is minimal for most of our cases studied. For the case of Python–REGEX, 28 sentences would be eliminated over three pattern instances for INPUT FORMAT OPTIONS. The threatened unsubstitutable sentences for this scenario generally provide a use case for the format option, such as "To match a literal '$', use \$ or enclose it inside a character class, as in [$]."[12] We also observed that the *Supporting Text* demonstrated the behaviour of the format option with an example such as that of | character: "Crow|Servo will match either 'Crow' or 'Servo', not 'Cro', a 'w' or an 'S', and 'ervo'."[12] As for the injection of additional sentences, automatic pattern reuse would result in a maximum of 11 additional sentences, hardly a documentation bloat.

4.3 Scattered Correspondences

As expected, some substitutable correspondences are not covered by the pattern. However this percentage varies widely across topic and programming language. We refer to matched substitutable sentences not covered by the pattern as *scattered correspondences*. Table 6 shows the distribution of scattered correspondences for different types of matches. Each cell includes the number of scattered correspondences of a given type, with the percentage with respect to the total number of correspondences with that match type. All of the 67 *identical* correspondences in Java–REGEX and the one such correspondence in Java–URL are covered by the pattern. In contrast, 15 of 20 (75%) of the *equivalent* correspondences in Java–REGEX are not covered by the pattern. We observe a general trend of more *scattered correspondences* with the decrease in the extent of the match.

However, the information reuse pattern we identified is only one, very systematic, way to reuse text between documentation types. Alternatives include referencing *prominent* (e.g., leading) sentences from the documentation directly. To determine the potential impact of this practice, we investigated to what extent *scattered correspondences* mapped to prominent sentences in the reference documentation that would be straightforward to link to.

Table 6: Distribution of sentences in tutorials not belonging to any instance of the information reuse pattern (*scattered correspondences*). Each cell includes the absolute number of correspondences as well as the the percentage with respect to the total number of sentences with that match type in the documentation. Hyphens (-) indicate there are no sentences at all for the correspondence type in the documentation.

Correspondence Type	**Java**			**Python**		
	`REGEX`	`URL`	`I/O`	`REGEX`	`URL`	`I/O`
Identical	0 (0%)	-	1 (100%)	0 (0%)	1 (100%)	-
Near-identical	2 (13%)	2 (100%)	6 (22%)	2 (22%)	-	-
Equivalent	15 (75%)	9 (45%)	66 (62%)	12 (37%)	10 (100%)	12 (100%)

Table 7: Distribution of locations of reference documentation sentences that belong to substitutable correspondences. *Scattered* refers to those sentences that are not covered by a pattern instance. *Total* indicates the total reference documentation sentences in substitutable correspondences. The percentage of sentences at a particular location per reference documentation are in parentheses.

Language	**Location**	**REGEX**		**URL**		**I/O**	
		Scattered	**All**	**Scattered**	**All**	**Scattered**	**Total**
Java	Description	0	54 (53%)	0	10 (45%)	0	10 (11%)
	Leading-A	0	21 (21%)	2 (18%)	2 (9%)	16 (22%)	51 (38%)
	Leading-B	0	0	0	1 (5%)	0	0
	Arbitrary	26 (100%)	27 (26%)	9 (82%)	9 (41%)	58 (78%)	68 (51%)
	Total	26 (100%)	102 (100%)	11 (100%)	22 (100%)	74 (100%)	129 (100%)
Python	Description	0	3 (5%)	0	0	0	0
	Leading-A	4 (20%)	14 (25%)	3 (27%)	3 (27%)	3 (25%)	3 (25%)
	Leading-B	1 (5%)	1 (2%)	1 (9%)	1 (9%)	1 (8%)	1 (8%)
	Arbitrary	15 (75%)	38 (68%)	7 (64%)	7 (64%)	8 (67%)	8 (67%)
	Total	20 (100%)	56 (100%)	11 (100%)	11 (100%)	12 (100%)	12 (100%)

For this purpose, we assigned each substitutable sentence to one of the four categories based on its location in the reference documentation and the context of its inclusion in the tutorial:

- **Description**: The sentence is a part of the entire description of an API element, and the entire description is present in the tutorial;
- **Leading-A**: The sentence is the leading sentence of an API element description;
- **Leading-B**: The sentence is the leading sentence of an explicitly defined block, such as a warning;
- **Arbitrary**: The sentence is at a location not covered by the above categories.

Table 7 shows the percentage distribution of the four categories above. We note that all correspondences of type *Description* are covered by the pattern. For the remaining (scattered) correspondences, this analysis shows to what extent it would be straightforward to refer to individual sentences in the reference documentation explicitly when constructing a tutorial, as opposed to copying and pasting them. Across our six cases, between 0 and 16 scattered sentences can be linked to prominent sentences of the reference documentation.

4.4 Discussion

We identified 38 instances of the information reuse pattern for four API element types in the six tutorials we studied, which varied across programming language and topic. These instances represent the systematic integration of reference documentation within a tutorial. They thus constitute preliminary evidence that generative technologies could help increase the level of automation that can support the creation of reference documentation.

The applications of the pattern we identified are not, however, completely systematic, as we observed some divergences between an exact correspondence between a pattern's structure and the content of the tutorial (see Table 5). These divergences can originate from one of two processes. On the one hand, the divergence can be *intentional*, if an author feels the structured information needs to be complemented by additional text. To accommodate for necessary variations, templating technology would require support for fine-grained customization. Determining the right level of granularity for content injection in tutorials is an interesting avenue for future work. On the other hand, divergences between pattern instances and tutorial text can also be *unintentional*, and either caused by an ad hoc approach to tutorial creation, or by software evolution that causes the tutorial text to become obsolete (Dagenais and Robillard, 2014). In this case, explicit use of information reuse patterns has great potential to help preserve the quality of the tutorial by limiting errors of omission.

We also observed that opportunities for information reuse beyond the systematic application of a pattern, with a few tutorial sentences in all but one case referring directly to prominent sentences in the reference documentation. The attribution of special status to certain sentences in natural language documents is not uncommon. For example, the Checkstyle tool can check that the first sentence of a Javadoc block ends with a period (jav, 2020); Git treats the first line of a commit message as the commit subject (git, 2020); the Wikipedia manual of style stresses the importance of an article's leading section (Wikipedia contributors, 2020), etc. With explicit support for linking to leading and other prominent sentences in reference documentation, we can expect this practice to gain adoption much beyond the tacit levels we observed in our case study.

Threats to Validity

It is possible that there exist information reuse patterns and pattern instances beyond the ones we have identified in our data set. Furthermore, it is possible to observe other patterns if our analysis was to be applied to tutorials beyond the six tutorials. Hence, we do not claim that the information reuse pattern we elicited is exhaustive. Similar to sentence correspondence, our observation of instances of reuse patterns is conservative in nature. The potential existence of other patterns and instances does not invalidate the results presented here. Any surfacing information correspondence pattern can be added to the the one we document here, as the patterns are mutually independent. Any occurrences of other instances would not alter our main outcome about the presence of recurring information correspondence patterns.

5 Related Work

Prior work has attempted to categorize the different types of software documentation and evaluate their quality (Section 5.1), assess the needs and preferences of users (Section 5.2), improve the quality of documentation (Section 5.3) and elicit patterns for documentation reuse (Section 5.4).

5.1 Documentation Types and Sets

Developers often draw upon multiple different resources while learning a new language. Parnin and Treude (2011) studied the type of information sources of the top ten web search results for JQuery API methods. While for 99.4% of methods tested, the official API documentation was in the top ten resources, other official documentation appeared only for 30.1% of methods. Blog posts were the next most frequently occurring at 87.9% of which about 49% were tutorials.

Watson (2012) developed a heuristic to evaluate whether API reference documentation contains important elements that help developers learn the features of a new API. At a broad level, Watson et al. (2013) evaluated documentation sets, i.e. the collection of different types of documentation for a software, based on *initial impression*, *experience* provided to a reader and any *additional data* that exists. They found that documentation components that developers prefer such as tutorials and sample applications were found in less than half of the 35 libraries studied.

Angelini (2018) studied the API documentation of eight web applications with the intention of better understanding technical writing patterns. All the web applications documentation sets studied contained at least one among an *Overview/Introduction*, a *Get Started*, a *Best practices/Usage guidelines* and a *Tutorial*, collectively referred to as *additional documentation*. However, no documentation set comprised the full set of supplementary additional documentation, confirming Watson's previous study. While standards exist for

documentation of computer software (Phoha, 1997; Ries, 1990; IEEE Standard, 2009), there is no such standards for structuring documentation sets, or collections of different documents, across languages and their APIs.

5.2 User Needs and Preferences of Documentation

Prior work to assess developer needs and preferences during development and maintenance has involved performing surveys and interviews. Work done by Robillard (2009), Robillard and Deline (2011), and Uddin and Robillard (2015) reiterates that among other preferences, developers would like more explained code examples, which are usually found in tutorials. Based on surveys of 25 participants, Garousi et al. (2013) concluded that, in industry, technical documentation (such as requirement specifications and design documents), source-code, communication with teammates and developers' existing knowledge are all approximately equally used during the development process. Josyula and Panamgipalli (2016) determined that for product architecture, learning new programming skills, and clarifying requirements, API reference documentation and online tutorials are frequently used information sources. Meng et al. (2018) observed that, when looking at a new API, their research participants asked "What can I do with this API" as their first question. This is also analogous to specific scenarios that tutorials and user guides can support. Meng et al. (2019) reported that participants spent 49% of their development time looking at the documentation, among which API reference documentation and cook book-like documentation are used nearly equally frequently. This confirms developers' multi-resource use, and calls for an analysis into the complementary nature between different types of documentation.

5.3 Assisting Developers and Improving Documentation

Various research projects have aimed to improve documentation, including work done to augment information from other resources such as source code (Kramer, 1999), make inferences from documentation that are less explicit (Zhong et al., 2009), or highlight directives in documentation (Dekel and Herbsleb, 2009). Rupakheti (2012) created *CriticAL (A Critic for APIs and Libraries)* that provides recommendations and descriptions for client code using the API. Treude et al. (2014) developed TaskNav, a tool for users to refer to a list of extracted tasks and how to perform them from textual documentation. Two developers rated the extracted tasks, resulting in 70% of them being meaningful to at least one of the two developers. Hence previous work shows that scenario-based instruction is an integral resource for developers, in addition to reference documentation.

Treude and Robillard (2016) took advantage of content similarity between software artifacts to use supervised machine learning techniques to identify and recommend *insight sentences* from Stack Overflow. Similarly, Jiang et al.

(2017) built a model to identify fragments from tutorials that are relevant to the corresponding APIs. In application, Caponi et al. (2018) created SSE, a tool which templates the reuse of documentation fragments for new HTML-like formatted documentation. Such work is based on the idea that information provided in different documentation on the same topic complement one another.

The most in line with the goals of our work, is that of Oumaziz et al. (2017) who studied the reuse of documentation tags in source code to generate reference documentation. They created a duplication detector to identify the duplicate documentation tags in seven Java APIs that use JavaDoc and reported that the most commonly duplicated tags are `param` and `throws` where 20% to 40% of these tags are duplicated. They determined that at least 57% of duplications were unintended "copy-pastes". They further proposed a simple documentation tag reuse mechanism to avoid duplicate information in documentation.

Surveys of 48 developers and managers conducted by Forward and Lethbridge (2002) revealed a number of interesting insights on the expectations of documentation by its users. One outcome, was that 82% of the participants agreed that tools must not only assist documentation creation, but also track changes in software to update and maintain documentation. Based on responses to their survey, they elicited a few high level requirements of documentation technologies, which include the need to support inter-documentation relationships to ensure consistency in documentation updation. Our work identifies this inter-documentation correspondence at the sentence level.

5.4 Patterns in Documentation

Dagenais and Robillard (2014) defined *documentation patterns* as *coherent sets of code elements that are documented together*. They proposed AdDoc, an automated method to capture these patterns in the documentation of frameworks and report inconsistencies between code and documentation during the development and maintenance processes of either artifact. They also create and evaluate a recommender system for changes in documentation based on AdDoc that intends to reduce the time and effort taken by documentation maintainers to keep track of all changes that would need to be made. This recommender was able to detect 99% of references in tutorials that pointed to deprecated or deleted code elements. Prior work has also attempted to bridge the gap between software clone detection and software documentation to identify and extract duplicate textual information in documents. These are useful in highlighting and mitigating redundancies and inconsistencies in documentation. Luciv et al. (2016), Koznov et al. (2015) and Luciv et al. (2018) proposed methods to automate the detection of repeated fragments of text in technical documentation. They also suggested methods to modify, refactor and manage the document based on the texts identified, to improve the quality of documentation (Koznov et al., 2017). These works perform duplicate detection

within a single documentation type and propose changes and refactoring on individual documents. On the other hand, we analyzed the correspondence of textual information *across* two documentation types and in our work, propose a pattern to reuse this information to aid documentation generation and promote consistency across documentation of APIs.

Previous related work has mainly focused on content and structure of Java API reference documentation. The only known prior work of direct comparison between documentation in Java and Python is conducted by Wildermann (2014) which reproduced and expanded on the work by Maalej and Robillard (2013) that identified knowledge types in Java API documentation. Fourney and Terry (2014) described the challenges presented when attempting to dissect tutorial-like material for automated understanding and processing. They found the need to formalize the content present in a tutorial with the purpose of templating online tutorials. While a number of work has focused on doing this in API documentation (Maalej and Robillard, 2013; Monperrus et al., 2012), tutorials seem to be far less explored, possibly because, as Fourney and Terry point out, even something as seemingly simple as determining what a *step* in the tutorial is, is a difficult problem.

In this work, we analyzed the information in tutorials with respect to the API reference documentation in both Java and Python. This would help better understand programming language documentation practices and the trend of commonly occurring API documentation and lack of sufficient tutorials, despite developers having voiced their needs for such materials. Our work augments previous cross programming language studies by providing insight into the extent of generalizability of relationships between documentation types in terms of their information content.

6 Conclusion

Our analysis of the correspondence of information between tutorials and reference documentation for API modules supporting regular expressions, URLs and Input/Output in Java and Python reveals that a large portion of the sentences (between 45% and 76%) contain API information in contrast to general supporting text. The sentences comprising *API Information* exhibit different level of correspondences to their possible sources in the API reference documentation, ranging from *identical* to *implied*. The percentages of occurrence of correspondence types vary across programming languages and API topics and do not follow any regularity throughout the tutorials. Our observation that 11% to 56% of the matched sentences can be directly substituted by the corresponding sentences in API reference prompts the potential for templating information reuse from one documentation type to another.

To this end, we defined a pattern for describing *systematic* documentation reuse in the tutorials. We found a total of 38 instances of this *information reuse pattern* in our case studies. This result supports the use of the elicited

pattern as a template during documentation generation. Moreover, such a template could address the divergence between content in different documentation types. For example, during our analysis, we discovered that the Java–URL tutorial [17] and reference documentation [27] provide inconsistent information for the resolution of a relative URL input to the `URL` constructor. Instead, a template for information reuse would effectively eliminate this kind of inconsistency.

In addition to this systematic reuse following a strict pattern, we also observed many instances where the information from the reference documentation found in the tutorial originated from a *prominent* location in the reference documentation. This practice indicates potential for better integration of document types through explicit linking between them.

Our corpus of correspondences between sentences in tutorials and reference documentation and the elicitation of the information reuse pattern can inform the development of advanced documentation creation technology. Such technology would mitigate the effort of documentation authors, potentially improve the quality of the documentation and provide familiarity for users when reading documentation of different APIs in different programming languages. This study, hence, provides a foundation towards a better understanding of the relationships between different documentation types in terms of information correspondence that could help bridge the gap between documentation content and the information needs of readers.

References to Documentation Sources

Below is the list of web URLs referenced in this paper. In the case of snippets of documentation used as examples, the corresponding URL defines the particular file in which the example text can be found.

[1] docs.oracle.com/javase/8/docs/api/java/net/ServerSocket.html
[2] docs.python.org/3/library/socket.html
[3] docs.oracle.com/javase/8/docs/api/java/util/regex/package-summary.html
[4] docs.oracle.com/javase/8/docs/api/java/net/package-summary.html
[5] docs.oracle.com/javase/8/docs/api/java/nio/file/package-summary.html
[6] docs.oracle.com/javase/tutorial/essential/regex/index.html
[7] docs.oracle.com/javase/tutorial/networking/urls/index.html
[8] docs.oracle.com/javase/tutorial/essential/io/index.html
[9] docs.python.org/3/library/re.html
[10] docs.python.org/3/library/urllib.html
[11] docs.python.org/3/library/functions.html
[12] docs.python.org/3/howto/regex.html
[13] docs.python.org/3/howto/urllib2.html
[14] docs.python.org/3/tutorial/inputoutput.html
[15] docs.oracle.com/javase/tutorial/essential/regex/intro.html
[16] docs.oracle.com/javase/tutorial/essential/regex/pattern.html
[17] docs.oracle.com/javase/tutorial/networking/urls/creatingUrls.html
[18] docs.oracle.com/javase/8/docs/api/java/util/regex/Pattern.html
[19] docs.oracle.com/javase/8/docs/api/java/net/URI.html
[20] docs.oracle.com/javase/tutorial/essential/regex/matcher.html

[21] docs.oracle.com/javase/8/docs/api/java/util/regex/Matcher.html
[22] docs.oracle.com/javase/tutorial/essential/io/fileAttr.html
[23] docs.oracle.com/javase/8/docs/api/java/nio/file/Files.html
[24] docs.oracle.com/javase/tutorial/networking/urls/urlInfo.html
[25] docs.oracle.com/javase/8/docs/api/java/net/URL.html
[26] docs.oracle.com/javase/tutorial/essential/io/file.html
[27] docs.python.org/3/library/json.html
[28] docs.oracle.com/javase/tutorial/networking/urls/readingURL.html
[29] docs.oracle.com/javase/tutorial/essential/regex/groups.html
[30] docs.oracle.com/javase/tutorial/essential/regex/unicode.html
[31] docs.oracle.com/javase/tutorial/essential/io/pathClass.html
[32] docs.oracle.com/javase/tutorial/essential/io/formatting.html
[33] docs.oracle.com/javase/tutorial/essential/io/walk.html
[34] docs.oracle.com/javase/8/docs/api/java/nio/file/FileVisitor.html
[35] docs.oracle.com/javase/8/docs/api/java/nio/file/FileVisitResult.html

References

(2020) Checkstyle - JavadocStyle. URL `https://checkstyle.sourceforge.io/config_javadoc.html#JavadocStyle`, [Online; accessed 07-May-2020]

(2020) Git-commit. URL `https://git-scm.com/docs/git-commit`, [Online; accessed 07-May-2020]

Al Omran FNA, Treude C (2017) Choosing an nlp library for analyzing software documentation: a systematic literature review and a series of experiments. In: 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), IEEE, pp 187–197

Angelini G (2018) Current Practices in Web API Documentation. In: European Academic Colloquium on Technical Communication, p 70

Caponi A, Di Iorio A, Vitali F, Alberti P, Scatá M (2018) Exploiting patterns and templates for technical documentation. In: Proceedings of the ACM Symposium on Document Engineering 2018, Association for Computing Machinery, New York, NY, USA, DocEng 18

Cleland-Huang J, Guo J (2014) Towards more intelligent trace retrieval algorithms. In: Proceedings of the 3rd International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering, pp 1–6

Dagenais B, Robillard MP (2010) Creating and Evolving Developer Documentation: Understanding the Decisions of Open Source Contributors. In: Proceedings of the 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp 127–136

Dagenais B, Robillard MP (2014) Using Traceability Links to Recommend Adaptive Changes for Documentation Evolution. IEEE Transactions on Software Engineering 11:1126–1146

Dekel U, Herbsleb JD (2009) Improving API Documentation Usability with Knowledge Pushing. In: Proceedings of the 31st International Conference on Software Engineering, pp 320–330

Forward A, Lethbridge TC (2002) The Relevance of Software Documentation, Tools and Technologies: A Survey. In: Proceedings of the ACM Symposium on Document Engineering, pp 26–33

Fourney A, Terry M (2014) Mining Online Software Tutorials: Challenges and Open Problems. In: Proceedings of Extended Abstracts on Human Factors in Computing Systems, pp 653–664

Garousi G, Garousi V, Moussavi M, Ruhe G, Smith B (2013) Evaluating Usage and Quality of Technical Software Documentation: An Empirical Study. In: Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering, pp 24–35

IEEE Standard (2009) Ieee standard for information technology–systems design–software design descriptions. IEEE STD 1016-2009 pp 1–35

Jiang H, Zhang J, Ren Z, Zhang T (2017) An Unsupervised Approach for Discovering Relevant Tutorial Fragments for APIs. In: Proceedings of the 39th International Conference on Software Engineering, pp 38–48

Josyula JRA, Panamgipalli SSSC (2016) Identifying the Information Needs and Sources of Software Practitioners: A Mixed Method Approach. Master's thesis, URL `http://urn.kb.se/resolve?urn=urn:nbn:se:bth-12832`

Koznov D, Luciv D, Basit HA, Lieh OE, Smirnov M (2015) Clone Detection in Reuse of Software Technical Documentation. In: Proceedings of International Andrei Ershov Memorial Conference on Perspectives of System Informatics, pp 170–185

Koznov D, Luciv D, Chernishev G (2017) Duplicate Management in Software Documentation Maintenance. In: Proceedings of the 5th International Conference on Actual Problems of System and Software Engineering. CEUR Workshops proceedings, vol 1989, pp 195–201

Kramer D (1999) API Documentation from Source Code Comments: A Case Study of Javadoc. In: Proceedings of the 17th Annual International Conference on Computer Documentation, pp 147–153

Krippendorff K (2018) Content Analysis: An Introduction to its Methodology. Sage Publications

Luciv D, Koznov D, Basit H, Terekhov A (2016) On Fuzzy Repetitions Detection in Documentation Reuse. In: Programming and Computer Software, vol 42, pp 216–224

Luciv D, Koznov D, Chernishev G, Terekhov A, Romanovsky KY, Grigoriev D (2018) Detecting Near Duplicates in Software Documentation. In: Programming and Computer Software, vol 44, pp 335–343

Maalej W, Robillard MP (2013) Patterns of Knowledge in API Reference Documentation. In: IEEE Transactions on Software Engineering, vol 39, pp 1264–1282

Meng M, Steinhardt S, Schubert A (2018) Application Programming Interface Documentation: What do Software Developers Want? In: Journal of Technical Writing and Communication, vol 48, pp 295–330

Meng M, Steinhardt S, Schubert A (2019) How Developers use API Documentation: An Observation Study. In: Communication Design Quarterly Review, vol 7, pp 40–49

Monperrus M, Eichberg M, Tekes E, Mezini M (2012) What Should Developers be Aware of? An Empirical Study on the Directives of API Documentation. In: Empirical Software Engineering, vol 17, pp 703–737

Oumaziz MA, Charpentier A, Falleri JR, Blanc X (2017) Documentation Reuse: Hot or Not? An Empirical Study. In: Proceedings of International Conference on Software Reuse, pp 12–27

Parnin C, Treude C (2011) Measuring API Documentation on the Web. In: Proceedings of the 2nd International Workshop on Web 2.0 for Software Engineering, pp 25–30

Phoha V (1997) A Standard for Software Documentation. In: Computer, vol 30, pp 97–98

Ries R (1990) IEEE Standard for Software User Documentation. In: International Conference on Professional Communication, Communication Across the Sea: North American and European Practices, pp 66–68

Robillard MP (2009) What Makes APIs Hard to Learn? Answers from Developers. In: IEEE software, vol 26, pp 27–34

Robillard MP, Deline R (2011) A Field Study of API Learning Obstacles. In: Empirical Software Engineering, vol 16, pp 703–732

Runeson P, Host M, Rainer A, Regnell B (2012) Case Study Research in Software Engineering: Guidelines and Examples. John Wiley & Sons

Rupakheti CR (2012) A critic for api client code using symbolic execution. PhD thesis, Clarkson University

Treude C, Robillard MP (2016) Augmenting API Documentation with Insights from Stack Overflow. In: Proceedings of 38th International Conference on Software Engineering, pp 392–403

Treude C, Robillard MP, Dagenais B (2014) Extracting Development Tasks to Navigate Software Documentation. In: IEEE Transactions on Software Engineering, vol 41, pp 565–581

Uddin G, Robillard MP (2015) How API Documentation Fails. In: IEEE Software, vol 32, pp 68–75

Watson R, Stamnes M, Jeannot-Schroeder J, Spyridakis JH (2013) API Documentation and Software Community Values: A Survey of Open-source API Documentation. In: Proceedings of the 31st ACM International Conference on Design of Communication, pp 165–174

Watson RB (2012) Development and Application of a Heuristic to Assess Trends in API Documentation. In: Proceedings of the 30th ACM International Conference on Design of Communication, pp 295–302

Wikipedia contributors (2020) Wikipedia: Manual of Style/Lead section. URL `https://en.wikipedia.org/wiki/Wikipedia:Manual_of_Style/Lead_section`, [Online; accessed 07-May-2020]

Wildermann S (2014) Messung der Informationstypen-Häufigkeiten in der Python-Dokumentation. Bachelor's thesis

Zhong H, Zhang L, Xie T, Mei H (2009) Inferring Resource Specifications from Natural Language API Documentation. In: Proceedings of the International Conference on Automated Software Engineering, pp 307–318

A Preprocessing Steps

In general, the following rules and preprocessing techniques for sentence extraction were adhered to:

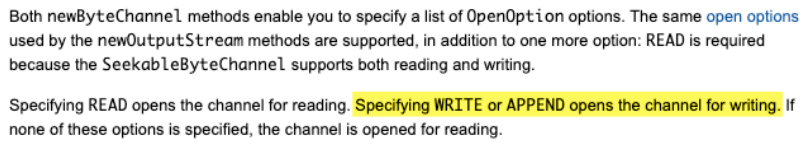
- Remove HTML tags *script*, *style*, *table*
- Insert a comma after the tokens 'e.g.' and 'i.e.'
- Insert a comma after the token 'etc.' if the word following this one began with a lower case alphabet.
- Replace multiple adjacent commas (occurring as a result of previous preprocessing steps) with a single comma.
- Replace newlines with spaces
- Replace multiple adjacent spaces with a single space
- Replace multiple adjacent periods (...) with a single period (.)
- In general, blockquotes, code blocks, images and the equivalents across the files were replaced by a single token *BLOCKQUOTE*, *CODE* and *IMAGE* respectively. These blocks were identified as being of a specific HTML tag type or having a specific HTML class.
- If a list item did not end in a period, the following item would be concatenated to the previous, separated by a semicolon.
- Finally, split on on a period followed by a space ('. ') and an exclamation followed by a space ('! ') to produce individual sentences

It is important to note here that inline HTML code tags in the sentence (*inline* and hence, did not involve line breaks) were maintained as is. Usually such pieces were names of the library or method being described. For example, "The `java.util.regex` package primarily consists of three classes: `Pattern`, `Matcher`, and `PatternSyntaxException`" [15].

B Reasons for *Similar* Correspondences

The reason for a two matched sentences to be considered *similar* but not *equivalent* could be one of the following:

- The sentence is a rephrased version of two non-neighbouring reference documentation sentences. As a result, these sentences cannot necessarily be systematically identified and merged without advanced mechanisms to merge the sentences coherently, efficiently and favorably for the reader in a human-like writing style. For example, consider the highlighted sentence in from `Java-I/O` tutorial [26].


Both newByteChannel methods enable you to specify a list of OpenOption options. The same open options used by the newOutputStream methods are supported, in addition to one more option: READ is required because the SeekableByteChannel supports both reading and writing.

Specifying READ opens the channel for reading. Specifying WRITE or APPEND opens the channel for writing. If none of these options is specified, the channel is opened for reading.

The description in the reference documentation of `newByteChannel` method to which it refers mentions this information in two separate non-consecutive sentences, as highlighted in the figure below [23].

The `options` parameter determines how the file is opened. The `READ` and `WRITE` options determine if the file should be opened for reading and/or writing. If neither option (or the `APPEND` option) is present then the file is opened for reading. By default reading or writing commence at the beginning of the file.

In the addition to `READ` and `WRITE`, the following options may be present:

Option	Description
`APPEND`	If this option is present then the file is opened for writing and each invocation of the channel's `write` method first advances the position to the end of the file and then writes the requested data. Whether the advancement of the position and the writing of the data are done in a single atomic operation is system-dependent and therefore unspecified. This option may not be used in conjunction with the `READ` or `TRUNCATE_EXISTING` options.

Combining these sentences to generate a coherent sentence as in the tutorial is beyond the scope of our work.

– The sentence references or is in conjunction with a specific example. Sentences that provide example-specific information are marked as *Supporting Text*. However, sentences that provide general information about the API in the context of an example are considered to have *similar* matches. `Python-I/O` tutorial [14] contains one such instance:

If you have an object `x`, you can view its JSON string representation with a simple line of code:

```
>>> import json
>>> json.dumps([1, 'simple', 'list'])
'[1, "simple", "list"]'
```

Our preprocessing steps result in the sentence extracted in the following format: "If you have an object x, you can view its JSON string representation with a simple line of code: CODE." [14]
The code snippet within this sentence as seen from the screenshot informs that the `JSON` string representation of an object x can be viewed using the `dumps` method. The description for the method in the reference documentation [27] states:

`json.`**`dumps`**`(`*obj, *, skipkeys=False, ensure_ascii=True, check_circular=True, allow_nan=True, cls=None, indent=None, separators=None, default=None, sort_keys=False, **kw*`)`

Serialize *obj* to a JSON formatted `str` using this conversion table.

Hence, replacing this tutorial sentence by its corresponding reference documentation sentence will result in a loss of the example which is integral for this sentence to provide useful information.

– It introduces a use-case for the reference API documentation. For example, in the `Java-URL` tutorial, the following sentence exists: "**After you've successfully created a URL, you can call** the URL's openStream() method to get a stream from which you can read the contents of the URL." [28] Here, the bold phrase describes when the `openStream` method can and should be used as opposed to the corresponding reference documentation that simply says: "Opens a connection to this URL and returns an InputStream for reading from that connection." [25], describing what the method performs.

– The matched API sentence may be providing excessive technical information. For example, the `Java-REGEX` tutorial states "The regular expression syntax in the java.util.regex API is most similar to that found in Perl." [15] On the other hand, the reference documentation goes into deeper details: "The Pattern engine performs traditional NFA-based matching

with ordered alternation as occurs in Perl 5." [18] A tutorial author might decide to omit technical details that the reference documentation contains which a reader referring to the tutorial would not be expected to benefit.

C Reasons for *Unmatched* Sentences

We describe the reasons for *unmatched* sentences in tutorials in detail below based on the ten categories listed in Table 4.

The majority of *unmatched* sentences in `Java-REGEX` provide information about the *underlying topic*, usually describing the general behaviour of the fundamental concept behind the API. The definition and description of a regular expression, its syntax, the behaviour of special characters or definitions of related terminology are examples of this category that we found in tutorial but not in the reference. For example, the following sentence defines a set of methods having similar functionality: "Capturing groups are a way to treat multiple characters as a single unit." [29]

We discovered that *usage*, i.e. general information on how an API is expected or intended to be used, is the most frequent category for *unmatched* in `Java-URL`, `Python-URL`, and `Java-I/O`. The `Java-URL` tutorial recommends how to handle a `MalformedURLException`: "Typically, you want to catch and handle this exception by embedding your URL constructor statements in a try/catch pair, like this: CODE." [17]

In `Python-REGEX`, the most commonly unmatched sentence category is that of *internal behaviour* with 30% of the sentences describing such information. For example, the following sentence was found in the tutorial, but not reference documentation: "Regular expression patterns are compiled into a series of bytecodes which are then executed by a matching engine written in C." [12]

A surprising 52% of sentences describe API *behaviour* in `Python-I/O`. It can be expected that descriptions of the way in which an API component performs is presented in the reference documentation, and so this finding is of interest. For example, the following sentence describes a particular behaviour of the `read` method on a file object: "If the end of the file has been reached, f.read() will return an empty string (''). " [14]

We found that sentences describing specific *use-cases* in which the API could be or is intended to be used, usually with the intention of motivating and justifying the usefulness of the API, were also not matched with reference documentation. This sentence from `Java-REGEX` is one such example: "The split method is a great tool for gathering the text that lies on either side of the pattern that's been matched." [16]

Sentences regarding *performance* of the API in terms of efficiency and scalability can also be observed in the tutorial, but their match could not be found in the reference documentation. The following sentence from `Python-I/O` is an example: "This is memory efficient, fast, and leads to simple code: CODE." [14]

We observed that some sentences providing *version information and backward compatibility* were not matched in the reference documentation. One example of a sentence providing information regarding content of a particular version is this sentence in `Java-REGEX`: "As of the JDK 7 release, Regular Expression pattern matching has expanded functionality to support Unicode 6.0." [30] This is a surprising finding because deprecation and enhancement information are generally specified in the reference documentation, in order to caution developers about no longer supported API components, or introduce them to new ones. We expect that this kind of information can be found in the version release notes and we leave the exploration of this documentation type to future work.

Some of our observations are unique to `Java-I/O` documentation. This, we theorize, is likely due to its large length and diverse range of sub-topics, providing greater scope for writing style variation. We found non-matched sentences providing *environment and platform specific information*, *API support* and *input configuration details* information only in this documentation. While describing the typical syntax of a file location, the documentation provides the following platform specific information: "In the Solaris OS, a Path uses the Solaris syntax (/home/joe/foo) and in Microsoft Windows, a Path uses the Windows syntax (C:\home\joe\foo)." [31] Another sentence describes whether a file system may be able to

support the API components provided: "A specific file system implementation might support only the basic file attribute view, or it may support several of these file attribute views." [22]

Sentences containing *input configuration details* information are ones which describe the structure of the input to an API. For example, in the `JAVA-IO` tutorial, the `width` is an element of the format specifier in the `format` API. The sentence provides the following information about `width`: "By default the value is left-padded with blanks." [32] The default behaviour of this element of the format specifier is not mentioned in the reference documentation.

We also identified one sentence describing a method for which the corresponding description in the API documentation was not descriptive enough to consider it as a match. While the tutorial states: "visitFile - Invoked on the file being visited." [33] The description of the `visitFile` method in the reference documentation is simply: "Invoked for a file in a directory." [34] While the sentences provide little explanation, the tutorial clarifies that this method is *invoked when a file is visited* as opposed to the reference documentation. We chose to treat this as an anomaly and not to categorize it as a separate *unmatched* category because of its low occurrence. Further, we found two more instances of descriptions in reference documentations that could have been matches for an tutorial sentence but were either incomplete or not clear in explanation. We consider both cases as *implied* matches because their meanings can be deduced given familiarity with the API. One example is shown below:

Tutorial:	**Reference documentation:**
CONTINUE - Indicates that the file walking should continue. [33]	public static final FileVisitResult CONTINUE Continue. [35]

It is important to note that these categories are not exclusive to *unmatched* sentences. There may be sentences which are matched to reference documentation that provide information on these categories. We leave the detailed comparison of documentation at the category level to future work.