Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

ARTICLE TYPE

Automatically Identifying Valid API Versions for Software Development Tutorials on the Web

Manziba Akanda Nishi*[1] | Kostadin Damevski[2]

[1]Department of Computer Science, Virginia Commonwealth University, Virginia, USA
[2]Department of Computer Science, Virginia Commonwealth University, Virginia, USA

Correspondence
*Manziba Akanda Nishi, Department of Computer Science, Virginia Commonwealth University, Richmond, VA USA. Email: nishima@vcu.edu

Present Address
Department of Computer Science, 601 West Main Street, Richmond, VA, 23284, USA

Summary

Online tutorials are a valuable source of community created information used by numerous developers to learn new APIs and techniques. Once written, tutorials are rarely actively curated and can become dated over time. Tutorials often reference APIs that change rapidly, and deprecated classes, methods and fields can render tutorials inapplicable to newer releases of the API. Newer tutorials may not be compatible with older APIs that are still in use.

In this paper, we first empirically study the tutorial versioning problem, confirming its presence in popular tutorials on the Web. We subsequently propose a technique, based on similar techniques in the literature, for automatically detecting the applicable API version ranges of tutorials, given access to the official API documentation they reference. The proposed technique identifies each API mention in a tutorial and maps the mention to the corresponding API element in the official documentation. The version of the tutorial is determined by combining the version ranges of all of the constituent API mentions. Our technique's precision varies from 61% to 89% and recall varies from 42% to 84% based on different levels of granularity of API mentions and different problem constraints. We observe API methods are the most challenging to accurately disambiguate due to method overloading. As the API mentions in tutorials are often redundant, and each mention of a specific API element commonly occurs several times in a tutorial, the distance of the predicted version range from the true version range is low; 3.61 on average for the tutorials in our sample.

KEYWORDS:
software development tutorials versioning, API versioning, API deprecation, software development tutorials

1 | INTRODUCTION

Software developers use various online resources like blogs, API documentation, mailing lists, tutorials, Q&A forums, e-books, etc., so that they can quickly learn new skills and techniques, expand knowledge which they have already obtained, or refresh their memory by remembering something they forgot [1,2]. They perform more than twenty software development related Web searches every day [3]. Xia et al. broke down developers' Web search queries by development phase, observing that one of the more frequent and difficult tasks for developers is to *search for usage examples or guidance on how to use third-party libraries/services* [4].

Online tutorials are a valuable source of knowledge for software developers who are learning to use development frameworks and API libraries/services or want to master a specific development technique. Tutorials provide a step-by-step narrative intertwined with examples, and, relative to most other sources of software documentation on the Web, provide a larger quantity of information that takes a significant amount of time to consume[5]. However, like other online software development documentation, most tutorials do not explicitly specify their prerequisites, i.e. what version of an API or framework tutorial is valid for. This causes problems both for very old tutorials (which may use outdated or deprecated APIs) and new tutorials (which use APIs that are not available for the developer's target platform).

In this paper, we posit that one of the problems with modern Web tutorials, among other similar documentation types, is that they may not work with the developers' target environment. We first motivate this problem by studying a corpus of tutorials for the Android platform. We report how long each tutorial tends to be valid for and what is most likely to cause a tutorial to become incompatible with new versions of a library or service. Next, we develop an automated technique to infer versions for online tutorials based on API documentation.

The primary source of the versioning problem in tutorials are mentions of API elements that have been deprecated and removed in the version of the library the developer is using, or mentions of API elements that have yet to appear. While APIs have significantly improved software development productivity, they are continuously changed in order to add new functionality or remove old and unnecessary functionality. In certain fast changing domains, APIs are modified quickly. For instance, each month, an average of 115 Android APIs are updated[6] and 3.6 APIs per month are deprecated[7]. Developers typically do not adopt new APIs rapidly, which is understandable because it requires additional effort and resources. For example, developers take a significant amount of time (almost 14 months on average) to update outdated APIs used in their software. Analyses show that 28% of Android API calls are not compatible with the latest released version and have a median lag time to update of 16 months.[6].

Although researchers have proposed approaches aimed at extracting the mentions of API elements from Web resources[8], all of these techniques fail to manage the existence of numerous API versions with removed or deprecated API elements[9]. Researchers have proposed tools that automatically detect the compatible versions of a source code repository, but these tools are not appropriate for detecting versions of documents like tutorials, which contain a combination of natural language text and source code[10]. Moreover, the code segments available in tutorials are not complete, and are often not compilable like the code segments present in a source code repository.

In this paper we propose a technique to determine the valid version range of the online tutorials. To the best of our knowledge, this is the first such attempt to deal with the versioning problem as it applies to online tutorials. The contributions of our work are following.

- empirical study of Android tutorials that confirms that version inadequacy is indeed a problem that exists for popular tutorials;
- technique to automatically determine the valid ranges of the tutorials by versioning their API mentions;
- evaluation of the technique using several different formulations of the classification task.

The remainder of this paper is organized as follows. Section 2 introduces the tutorial versioning problem and presents empirical results to support its existence. We describe our technique for automatically determining the valid version ranges of tutorials in Section 3. We apply the technique to our dataset of manually annotated tutorials and provide an evaluation in Section 4. In Section 5, we describe relevant related research and conclude this paper in Section 6.

2 | MOTIVATION

Developers commonly use online tutorials to study new or unfamiliar APIs, spending considerable amount of time to follow a tutorial step-by-step or to decide exactly which parts of a tutorial are relevant to their ongoing tasks[11,12,13]. Expert developers share their procedural or "how to" knowledge, creating new tutorials that support various development activities. Tutorials targeting software development can use various media, taking the form of written documents, interactive programs, or screen recordings[11]. In this research we only consider written online tutorials available on the Web.

Researchers have studied the effects of API deprecation and removal in various development ecosystems, leveraging online resources like Stack Overflow and GitHub[14,15,16]. For instance, researchers have examined why APIs change[17] and how API

changes affect developers[18]. Researchers studying whether changes in APIs trigger StackOverflow questions detected a strong increase in the number of questions asked about frequently changed API methods[19]. Studies have also found that when developers ask questions about new APIs they get more answers, but high quality answers take significant time to accumulate[20]. We were not able to find any empirical studies of API deprecation or removal that examined the effect of this phenomenon specifically in software development tutorials.

Software development tutorials commonly contain both natural language descriptions and code segments, which provide examples to further clarify the narrative. Often, tutorials mention APIs in the description and in code segments. Typical code segments consist of several lines of codes, in some cases consisting of complete or nearly complete classes or methods. When tutorials mention one or more APIs that have been removed or deprecated, then the entire tutorial is no longer valid for the current version of the API as many developers read these resources in their entirety. Such tutorials often can give incorrect information to developers that use newer (or older) APIs leading to loss of development productivity. In most cases, tutorials do not specifically express or warn about version compatibility. In APIs with poor forward and backward compatibility, tutorials quickly grow outdated. The aim of our research is to automatically detect the API versions of tutorials, as it is very time consuming or even impossible to manually check every API of the tutorial to determine their version compatibility.

Figure 1 shows a tutorial[21] that discusses Android's `startActivityForResult` API method, which is one of the most commonly used API methods in Android app development. Although the tutorial authors inform their readers that there are two variants of this method, there is no indication of the applicable API version number. In fact, the first variant `startActivityForResult (Intent intent, int requestCode)` was added in API level 1, while the other `startActivityForResult (Intent intent, int requestCode, Bundle options)` was added much later, in API level 16. Both APIs are still in use. However, the tutorial reader can easily be confused by this discrepancy in the valid API versions of these two similar methods, e.g. resulting in difficulties when maintaining legacy applications.

A complementary motivating example is in the tutorial in Figure 2, where the tutorial shows an example code segment that utilizes the `FloatMath` class (on line 22). This class was deprecated in API level 22 and removed in level 23. While this class is not the focal point of this tutorial, reusing the code snippet that references it can lead to difficulty and waste time looking for alternative classes or tutorials. The tutorial prominently displays that it was written in 2013 and has not been updated since, which is perhaps an indicator of dated information, but there are numerous cases where a tutorial's age is not so easy to discern.

If the developers are informed of corresponding versions of APIs mentioned in tutorial and the version range in which the tutorial is active or working, then they can make quick decisions whether the tutorials are worth reading and whether code segment examples of this tutorial are worth reusing. In some cases developers may be willing to adjust the API level of applications to fit the tutorial, but perhaps in the majority of cases developers would search for a different resource to learn from. In yet other cases, developers are specifically interested in learning which APIs have been removed or deprecated, information that would also be available to readers if tutorials published their valid API levels.

2.1 | Empirical Study

To understand the scope of the tutorial versioning problem, we conducted an empirical study using Android tutorials available on the Web. We selected Android as its APIs have exhibited a lot of churn in recent years. Table 1 shows the release dates of each Android version. New releases are frequent and API changes, including additions, deprecations and removals, are considerable between pairs of releases. New Android APIs are often added to support new features that are to be available in new types of mobile devices or to support enhancements in the OS or runtime. However, as most devices available in the marketplace are older, to reach the widest audience applications have to be able to support older version of the API.

The lifecycle of an API is shown in Figure 3. After a new API class method or field is added, it can go through three separate lifecycle stages: an API can continue to be valid, be deprecated, or removed. Deprecation is a stage intended to warn developers that an API element will be soon removed. Based on the lifecycle, for each API we can associate an added version, removed version and depreciated version. To illustrate this point, Table 1 also shows the number of removed classes and number of deprecated classes in each version of Android. In version 5.0-5.1.1, the highest number of classes (400) were deprecated and in version 4.1-4.3.1 highest number of classes were removed. In this paper, we used the listing of Android APIs provided by the Android Official Documentation[23], Android API Differences Report[24], and Android Support Library API Differences Report[25].

The goal of our study is to determine if tutorials available on the Web may suffer from the problem of being only applicable to specific versions of APIs. To measure this, we collected a large set of Android tutorials from several different sources using stratified sampling based on 1) the source of the tutorial; and 2) the published year of the tutorial. We ended up with 13 tutorials

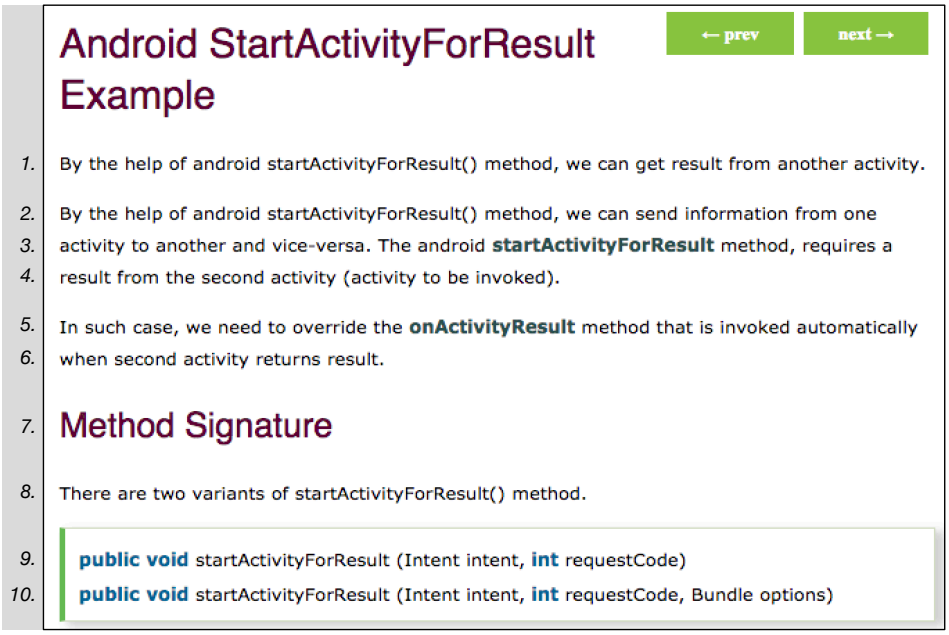

FIGURE 1 Portion of a tutorial discussing the `startActivityForResult` Android API[21]

Code Name	Version Number	Initial Release	API Level	Number of Removed Classes	Number of Deprecated Classes
Base	1.0	10/2008	1	-	-
Base	1.1	02/2009	2	-	-
Cupcake	1.5	04/2009	3	0	5
Donut	1.6	09/2009	4	2	4
Eclair	2.0-2.1	10/2009	5-7	0	39
Froyo	2.2-2.2.3	05/2010	8	0	1
Gingerbread	2.3-2.3.7	12/2010	9-10	9	0
Honeycomb	3.0-3.2.6	02/2011	11-13	6	10
Ice Cream Sandwich	4.0-4.0.4	10/2011	14-15	0	5
Jelly Bean	4.1-4.3.1	07/2012	16-18	38	43
KitKat	4.4-4.4.4	10/2013	19-20	0	2
Lollipop	5.0-5.1.1	11/2014	21-22	0	400
Marshmallow	6.0-6.0.1	10/2015	23	36	7
Nougat	7.0-7.1.2	08/2016	24-25	3	37
Oreo	8.0	08/2017	26-27	4	12
Android P[18]	9	07/2018	28	9	41

TABLE 1 The set of different Android versions, corresponding API levels, and number of removed and deprecated classes.[26 23 24]

that spanned 4 sources and 5 years (2013 - 2017). The year of publishing of the tutorial is usually reported by the tutorial itself. We selected tutorials that had prominently displayed modification dates that appeared consistent. However, in one tutorial, we still observed some minor errors in the reported dates, which we were able to manually correct based on related context.

The task of determining the valid API version for a specific tutorial can be reduced to determining the version of all the API mentions in each tutorial. To this end, we created a list of tokens that contains all the API names (i.e. methods, classes,

FIGURE 2 Portion of a tutorial utilizing the deprecated API `android.util.FloatMath`[22].

fields, packages) by parsing the Android Official Documentation[23]. Using this list of Android API tokens we performed string matching with all of the text (including source code and natural language) in the 13 tutorials in order to extract all of the potential API mentions. We ignored the text fonts and formatting, as these were specific to each tutorial. The matched tokens contained numerous false positives. Next, we manually annotated each matched mention in order to validate whether it is truly an Android API that is being referred to, and, if so, determine its version.

2.1.1 | Manual Annotation Procedure

For the task of determining, if potential API mentions in the 13 tutorials are valid and determining the exact API class, method, or field, that is being referred to, we recruited two Ph.D. and two M.S. students, who had taken a course on Android and had several years of Java experience. We instructed the annotators to begin by reading through the whole tutorial in order to get a good grasp on the context. Following this, they were to examine each potential mention, focusing on the text and code context surrounding it, and determine whether this is truly a mention or just a spurious match. For each validated mention the annotators were to identify the specific API element, by identifying the corresponding URL in the latest Android documentation.

During the annotation, there were mentions where a method name or field name has different variants. For instance, multiple methods with same name can have different parameters and return types, while mentions with same name can belong to different classes or packages. The annotators were asked to pay special attention to these cases, and to carefully disambiguate them. Since the annotators only used the latest version of the API, there were cases where they were not able to locate the appropriate API for a mention in the tutorial. For these cases, after annotators completed the annotation procedure, the authors checked over the difficult to disambiguate cases in order to confirm they were correctly annotated and corrected the errors.

Tutorial Title & Source Website	Year Published	Ratio of Valid Mentions	Valid Version Range
Learning to Parse XML Data in Your Android App – sitepoint.com –	6/7/2013	36%	[1-28]
Navigation Drawer Android Example – stacktips.com –	10/16/2013	35%	[22-27]
How to Get all Registered Email Accounts in Android – stacktips.com –	4/7/2014	40%	[5-22]
Scheduling Background Tasks in Android – sitepoint.com –	4/30/2014	37%	[3-28]
Android Lollipop Swipe to Refresh Example – stacktips.com –	1/27/2015	33%	[22-28]
Android Navigation Drawer for Sliding Menu / Sidebar – androidtutorialpoint.com –	12/15/2015	32%	[25-28]
Building Android applications with Gradle - Tutorial – vogella.com –	4/18/2016	10%	[1-28]
Android Facebook Login Tutorial – Integrating Facebook SDK 4 – androidtutorialpoint.com –	5/2/2016	37%	[26-28]
Using ViewPager to Create a Sliding Screen UI in Android – sitepoint.com –	8/31/2016	45%	[25-28]
Retrofit, a Simple HTTP Client for Android and Java – sitepoint.com –	1/11/2017	27%	[25-28]
Convert Speech to Text in Android Application – stacktips.com –	1/30/2017	36%	[26-28]
Android Chat Bubble Layout with 9 patch Image using ListView – androidtutorialpoint.com –	3/22/2017	30%	[25-28]
Understanding Androids Parcelable - Tutorial – vogella.com –	4/20/2017	42%	[1-28]

TABLE 2 Range of valid versions of the tutorials in our set.

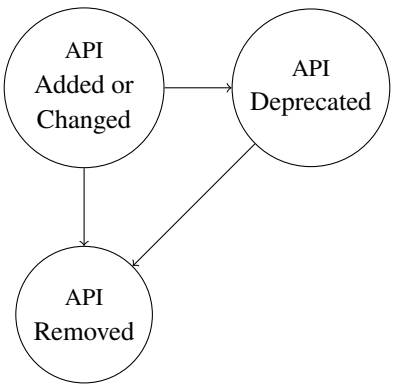

FIGURE 3 Lifecycle of an API element.

2.1.2 | Analysis of Findings

We present an overview of the findings of the empirical study, including the title, source, published data, ratio of valid mentions, and the valid version range of the 13 tutorials in Table 2. We observe a variety of valid version ranges for the tutorials. There is no observable consistency in the API-levels used in the tutorials based on the publication years, tutorial sources, or the topics discussed in the tutorials. In 3/13 tutorials, the version range encompasses all currently known releases of the Android API, but many others are significantly constrained; 8/13 lack full backward compatibility and 2/13 lack both full forward and full backward compatibility. We also observe in Table 2 that the ratio of the valid mentions varies from 10% to 45%, never approaching high

proportions for any of our tutorials. This indicates that finding out API names from tutorials is more challenging than simple string matching because almost half of the tokens that are matched to API names are not actual API mentions.

2.1.3 | Threats to Validity

Our empirical study suffers from several threats to its validity. One threat is in the correctness of the manual annotations. This risk is mitigated by the fact that the annotators possess sufficient skills in Android and Java, as well as by the fact that the authors manually re-checked the correctness of difficult annotations. Another threat to validity is in the selection of tutorials for the study. We used stratified sampling based on two features. However, we only randomly sampled a single tutorial for a feature pair, which could have lead to bias. There also may have been other relevant tutorial features that we did not consider in our sampling.

3 | AUTOMATED VERSIONING OF SOFTWARE DEVELOPMENT TUTORIALS

By automatically determining the version of a tutorial, from a set of official documentation for the APIs that the tutorial is describing, we could convey information that developers would be able to use to quickly determine whether a tutorial is compatible with their environment. Each tutorial contains a set of terms, located in code examples and in the surrounding narrative, that match API element names. Determining the version of these terms is required in order to determine the version of the tutorial. However, some of these matches are spurious and should be ignored. For instance, a API method named `run()` would produce a false positive match with a sentence in the tutorial narrative that uses the verb run. Therefore, extracting the range of versions for a tutorial is a two step process:

1. Differentiate valid from spurious API mentions in the tutorial.
2. Uniquely map each valid API mention to its exact API element (i.e. class, field or method).

Once we disambiguate and resolve the valid API mentions, it is pretty straightforward to extract the corresponding API version number of the mention and subsequently the entire tutorial. In this paper we apply natural language and machine learning based methodology that can identify valid mentions of the tutorial as well as disambiguate among the multiple occurrences of the APIs which have same name but have different signatures.

3.1 | Versioning Workflow

Figure 4 shows the workflow of our approach for automated tutorial versioning. The input to our technique is (1) official API documentation for each API version and a (2) tutorial whose version range we would like to determine. There is no standard way to obtain API documentation, so we developed an Android specific parser that extracts API information from the Android Official Documentation[23], Android API Differences Report[24], and the Android Support Library API Differences Report[25] available on the Web. Our parser extracts the signature and summary of all the added, removed and deprecated API elements. We store the parsed documentation in the form of a database for convenient access.

In the next step of the workflow, we tokenize the tutorial using white spaces, remove all punctuation, and perform simple string matching of the tutorial's tokens to the API element names of the Android Official Documentation[23]. The matching tokens constitute a list of candidate API mentions in the tutorial. For each of the candidate mentions we also obtain the context, consisting of a bag of words representation of the surrounding lines of text in the tutorial. We use a threshold of one line before and after the candidate mention, including the line where the candidate mention occurs.

Each candidate mention can have one or more potential API elements that it can be mapped to. Our task is to disambiguate which API the mention truly refers to. However, it is also possible that the candidate mention is only spuriously matching an API element - i.e. it is an invalid mention. For the disambiguation task, we compute a set of four features that highlight natural language and source code aspects of each mention, and use these features as input to a classifier. Once the classifier maps the candidate mention to its correct API, or marks it as not a valid mention, we aggregate the versions of all the mentions in a tutorial to determine that tutorial's valid API version range. To find out the valid version range of the tutorials, it is important to detect whether tutorial contains APIs which are already removed or deprecated and its corresponding removed or deprecated versions. To detect removed or deprecated APIs, we have used Android API Differences Report[24], and Android Support Library API Differences Report[25].

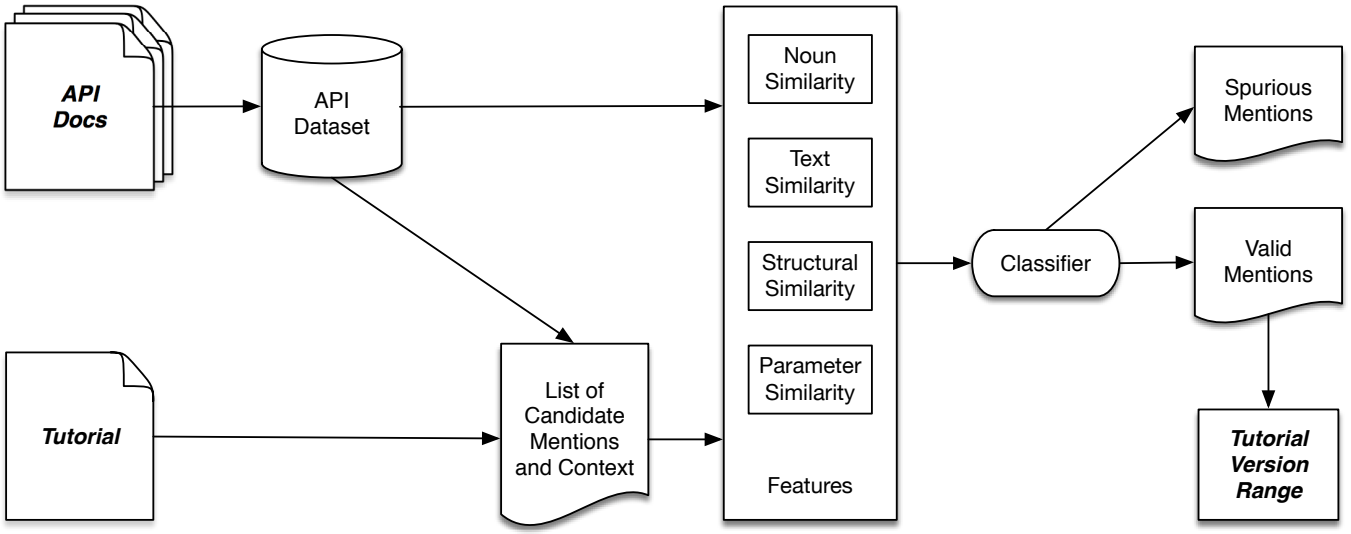

FIGURE 4 Overview of our technique for automated versioning of software development tutorials.

To calculate the valid version range of a tutorial using all of the disambiguated mentions in the tutorial, we apply the following rules:

1. If there are no mention of a removed or deprecated APIs in the tutorial, the valid version of the tutorial ranges from the most recent version when all of the mentions were present in the API to the most recent available version of the API.
2. If there is only one removed or deprecated API mention in the tutorial, the valid version ranges from the most recent version when all of the mentions were present in the API up to the version prior to the API removal.
3. If there are multiple removed or deprecated API mentions in the tutorial, the valid version ranges from the most recent version when all of the mentions were present up to one version prior to the minimum of all the versions of the removed/deprecated APIs that were mentioned.

3.2 | Features

We selected four features to extract and uniquely characterize candidate API mentions that occur in tutorials. These are: (1) Noun Similarity, computed between the candidate API mention's context (from the tutorial) and an API element's description (from the API documentation), (2) Text Similarity, computed using a vector space model (i.e. tf-idf) of the mention's context and API document description, (3) Structural Similarity, computed between the mention's context and structurally related members of the API; and (4) Parameter Similarity, computed directly between the candidate mention and API element. In the following, we discuss each of these features in more detail and use an example – a mention to API element pair from Figure 1 – to illustrate each feature.

3.2.1 | Noun Similarity

The motivation behind selecting the Noun Similarity metric is that the noun words in the surrounding context play an important role in identifying tutorial mentions that describe a specific API[8]. To be clear, as nouns we also consider the names of classes, packages, fields and methods. Our hypothesis is that nouns that occur in the description of the API in the Android Official Documentation tend to appear more in surrounding context of corresponding mentions in Android tutorials. We calculate the Noun Similarity measure using Jaccard similarity of the nouns in the bag of word context captured for each mention and the nouns occurring in the description of the candidate APIs from the official documentation:

$$NounSimilarity(m, API) = \frac{|nouns(m_{context}) \cap nouns(API_{desc})|}{|nouns(m_{context}) \cup nouns(API_{desc})|}$$

,where $m_{context}$ is the surrounding tokens of each mention (i.e. the context) and API_{desc} is the descriptive text for each API element that can be found in the official documentation.

As an example, consider the potential mention-API element pair of: 1) the `startActivityForResult` mention on line 1 of Figure 1 and 2) the API method `startActivityForResult(Intent intent, int requestCode)` in the Android Official Documentation. The Noun Similarity measure computes the similarity in the nouns occurring in the mention's context in the tutorial, e.g., `['help', 'android', 'method', 'result', 'activity', 'information', ...]` and the nouns occurring in the description of this method in the API documentation, e.g., `[['Intent', 'int', 'Bundle', 'options']`.

3.2.2 | Text Similarity

Noun Similarity focuses on one aspect of aligning the natural language context of an API mentioned in a tutorial and its textual description in the official documentation. In order to capture the influence on any remaining, yet important, terms, we use the Text Similarity metric, which first computes the term frequency - inverse document frequency (tf-idf) score of each matching term, and then computes the sum, as follows:

$$TextSimilarity(m, API) = \sum_{t \in (terms(m_{context}) \cap terms(API_{desc}))} tf(t) * idf(t)$$

,where $m_{context}$ is the surrounding tokens of each mention (i.e. the context) and API_{desc} is the descriptive text for each API element that can be found in the official documentation.

For the potential mention-API pair of the `startActivityForResult` mention on line 1 of Figure 1 and `startActivityForResult(Intent intent, int requestCode)` the Text Similarity measure computes the similarity among all of the terms in the mention's context (as can be observed in Figure 1) and the description of the `startActivityForResult(Intent intent, int requestCode)` API method, which is as follows: *"Same as calling startActivityForResult(android.content.Intent, int, android.os.Bundle) with no options."*

3.2.3 | Structural Similarity

Some API elements have common names, increasing the number of potential APIs in the official documentation for each candidate mention in the tutorial. In order to help disambiguate these cases, we introduce a metric based on program structure, which often works when tutorial authors include snippets of code. Using the code context of a mention, we can extract related package and class names and try to match them in the API documentation.

For instance, in the tutorial shown in Figure 1, the `android.app.Activity` class is imported, which can provide a hint that the `startActivityForResult` method belongs to `android.app.Activity` class. Although the effect of this metric can be strong, there are many cases where sufficient hints are not available in the surrounding code.

To compute Structural Similarity, similar to Uddin et al.[8], we use an island parser to process the surrounding code segments of a mention in order to identify either fully qualified or unqualified names of variable types. Different from the other metrics, here we look more broadly, using several lines in the tutorial text, for surrounding code snippets. In the found code snippets, we extract the types using import statements, class declarations, and interfaces or extended classes. We use this list of types to compare to related types in the official API documentation. For the API documentation we gather the entire type hierarchy of the class (or of the containing class for a candidate field or method). A definition of Structural Similarity is as follows:

$$StructuralSimilarity(m, API) = \frac{|types(m_{bigcontext}) \cap types(API)|}{|types(m_{bigcontext})|}$$

,where $m_{bigcontext}$ is the larger set of surrounding tokens of each mention, while $typesc$ are the set of type names in the API hierarchy, as specified in the official documentation, or encountered in a code segment found in the tutorial mention's surrounding context.

Since the mention to `startActivityForResult` on line 1 of Figure 1 has no source code in its vicinity, this Structural Similarity will be zero.

3.2.4 | Parameter Similarity

Methods are the most common API element we encounter in tutorials. A method's parameters can be a crucial feature in disambiguating multiple methods with same name. Method overloading is supported in many languages and is a common pattern

found in many APIs. Considering again the mention of `startActivityForResult` in Figure 1 we observe that both variants of this method are part of the `Activity` class. The different parameter number and types are the only aspect distinguishing the two APIs being referred to by this mention, and for this purpose we introduce the Parameter Similarity feature.

Parameter Similarity is computed between the method parameters of a mention in the tutorial and those of the candidate APIs in the official documentation. For efficiency, in matching, we first consider the number of parameters and then we attempt to match their types, assuming that type information is available. The metric is binary, producing a value of 1 if the parameters of the API in the two sources match, and 0 otherwise. A definition of this feature is as follows:

$$ParameterSimilarity(m, API) = \begin{cases} 1, & \text{if } types(parameters(m_{context})) = types(parameters(API)) \\ 0, & \text{otherwise} \end{cases}$$

,where $types(parametersc)$ specifies the types of the parameters found in method definitions in the API or method invocations in the tutorial.

As for the Structural Similarity, when a mention has no source code in the surrounding text, as `startActivityForResult` on line 1 of Figure 1, its Parameter Similarity is also zero.

4 | EXPERIMENTAL STUDY

Each tutorial contains a number of potential API mentions. Each of these API mentions needs to be disambiguated and mapped to the exact API member it corresponds to, or marked as a spurious match that is not a true mention. Subsequently, to find out the range of versions of a tutorial we can compute the valid ranges for the constituent API mentions. The aim of our experimental study are the following research questions:

- **RQ 1:** Does our technique for automatic versioning of software development-related tutorials accurately map mentions to their corresponding APIs?
- **RQ 2:** Is our technique for automatic versioning of software development-related tutorials effective at determining valid version ranges?

The effectiveness of our technique in **RQ2** is dependent on achieving a reasonable accuracy on **RQ1**.

4.1 | Experimental Setup

In this section we describe the method and metrics we use to evaluate the research questions. Our evaluation dataset is the one constructed with the empirical study described in Section 2.1. It consists of 13 tutorials sampled via stratified sampling with 75879 potential API mentions and the Android API documentation scraped from official Web pages. From these, 1268 are actual API mentions consisting of 744 classes, 54 fields and 470 methods. We used Python to implement our technique, leveraging the popular natural language processing libraries NLTK[27] and TextBlob[28]. For island parsing of the code snippets embedded in tutorials we used the SrcML parser[29].

The versions when API mentions in our tutorial dataset were initially introduced in Android follow the distribution shown in Figure 5. A few Android releases (e.g. 1, 11, and 22) introduced popular API members that are commonly referenced in our tutorials. However, many other Android releases mostly contributed additional or specialized functionality referenced by few (or no) mentions in our tutorial set. Therefore, the problem of determining which API versions are supported by a tutorial is skewed by commonly occurring API elements.

The constituent problem of matching a mention to an API element (i.e. a specific class, method or field) is naturally formulated as binary classification. That is, for each potential mention in a tutorial we consider a binary decision of whether it belongs to each, out of sometimes several, possible APIs with the same name. This task is also heavily skewed by common API names, for which the classification task is significantly more difficult. For instance, the `toString()` API method occurs in 669 different Android classes. On the other hand, the `replacement()` method occurs in only 2 different classes, `CharsetEncoder` and `CharsetDecoder`. So some mentions in our dataset are very hard to disambiguate, while others are straightforward. To measure our approach's effectiveness and answer RQ1, we use metrics common in binary classification problems.

Tutorial Name	Precision	Recall	F1-score
Learning to Parse XML Data in Your Android App	41%	31%	35%
— Class	82%	74%	78%
— Field	100%	100%	100%
— Method	83%	24%	37%
Navigation Drawer Android Example	29%	48%	36%
— Class	58%	80%	67%
— Field	–	–	–
— Method	66%	32%	43%
How to Get all Registered Email Accounts in Android	85%	55%	67%
— Class	76%	100%	86%
— Field	100%	100%	100%
— Method	85%	50%	62%
Scheduling Background Tasks in Android	53%	32%	40%
— Class	84%	76%	80%
— Field	90%	60%	72%
— Method	100%	42%	60%
Android Lollipop Swipe to Refresh Example	17%	67%	27%
— Class	67%	89%	76%
— Field	–	–	–
— Method	27%	61%	32%
Android Navigation Drawer – for Sliding Menu / Sidebar	25%	44%	32%
— Class	58%	83%	68%
— Field	100%	100%	100%
— Method	80%	25%	38%
Building Android applications with Gradle - Tutorial	12%	29%	17%
— Class	61%	100%	75%
— Field	100%	100%	100%
— Method	57%	88%	69%
Android Facebook Login Tutorial - Integrating Facebook SDK 4	40%	26%	32%
— Class	69%	87%	77%
— Field	85%	100%	91%
— Method	34%	30%	32%
Using ViewPager to Create a Sliding Screen UI in Android	54%	39%	45%
— Class	77%	100%	87%
— Field	–	–	–
— Method	40%	20%	27%
Retrofit, a Simple HTTP Client for Android and Java	14%	28%	19%
— Class	75%	93%	83%
— Field	50%	66%	57%
— Method	29%	68%	41%
Convert Speech to Text in Android Application	12 %	19%	15%
— Class	80%	43%	56%
— Field	83%	83%	83%
— Method	66%	13%	22%
Android Chat Bubble Layout - with 9 patch Image using ListView	70%	46%	55%
— Class	79%	84%	81%
— Field	100%	50%	66%
— Method	59%	39%	47%
Understanding Androids Parcelable - Tutorial	60%	40%	48%
— Class	95%	72%	82%
— Field	–	–	–
— Method	76%	66%	71%
Average Combined	39%	38%	36%
— **Average Class**	74%	82%	76%
— **Average Field**	89%	84%	85%
— **Average Method**	61%	42%	44%

TABLE 3 Results using tutorials as units. (-These tutorials have no positive fields.)

- Precision – measures the ability of a classifier in labeling positive samples as positive and avoid labelling positive samples as negative [30]. High precision indicates that our technique does not misclassify mentions as wrong APIs, or classify spurious mentions as mentions. In other words, precision is a good metric to evaluate a model when the cost for false

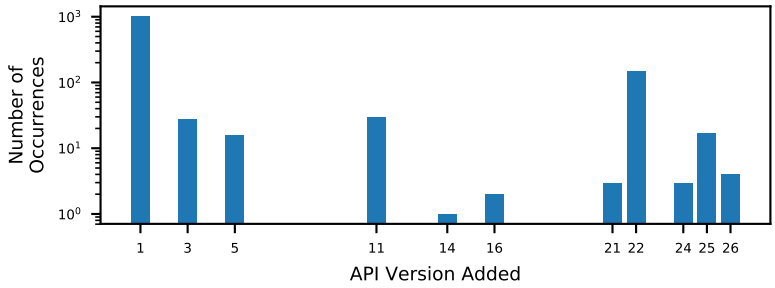

FIGURE 5 The distribution of the added versions across all the tutorial (log scale).

positives is high.[31]. Downstream, low precision could result in our technique selecting an overly restrictive version range for a tutorial.

- Recall – measures a classifier's ability to find all the positive samples[30]. High recall is indicative of our technique's ability to recognize all of the API mentions in a tutorial, missing few or none. Recall is used to evaluate a model when the cost of false negatives is high[31]. Downstream, low recall could result in our technique not being restrictive enough in versioning tutorials.

- F1-Score (binary) – is a popular metric that combines precision and recall. We compute the binary F1-score, which is applicable in problems like ours when the predicted or target class is binary[32].

For **RQ2**, we require a metric that contrasts the true version range of a tutorial, as determined using manual annotation, from the set of versions of the set of disambiguated mentions in that tutorial. Therefore, for comparing two version ranges we use the following metric:

- Manhattan Distance – measures the distance between two points as the sum of the absolute differences of their Cartesian coordinates[33]. We use the Manhattan Distance to assess the difference between the predicted and actual version range of a tutorial, as the measure follows intuitive notions of a distance between two version ranges. For instance, it penalizes errors in underestimating and overestimating the range equally.

For this supervised learning problem, we explore two different train-test split strategies: (1) using mentions as a unit; and (2) using tutorials as a unit. In the first strategy a randomly chosen portion of mentions is used in the training set and the remainder constitutes the test set. Likely this results in some portion of the mentions of each tutorial (from our set of 13) to be placed in each set. In the second strategy, we examine using all of the mentions from a portion of the tutorials as the training set, leaving the remaining whole tutorials as the test set. The second strategy is meant to convey a more realistic deployment of our technique, where the trained classifier has not seen any of the mentions of the new, previously unseen, tutorial whose version it determines.

Classification Algorithms. For the binary classification problem we choose the Random Forest Classifier, which has been shown to produce good results on a variety of problems[34]. We use the default settings for the Random Forest Classifier: numTrees = 1500. In addition, in order to better address the problematic popular mentions, where numerous possible API matches exist (e.g. `toString()`) for a mention in a tutorial, we reformulate the classification task as multi-instance classification. In multi-instance classification[35], there are multiple instances that are grouped together in a bag, and the algorithm's task is to predict the label of the bag taking into account all of the instances that comprise it. That is, the classifier first predicts whether a bag corresponds to a spurious vs. non-spurious mention. Subsequently, if it is a real mention, the highest expressed instance within the bag should correspond to the specific API we map the mention to.

To make this clearer, consider a case where the candidate mentions in the tutorial are denoted as $m_1, m_2, m_3, ..., m_n$. Each of these mentions has a set of potential candidate APIs $c_1, c_2, c_3, ..., c_k$, extracted from the official API documentation. In our classification task, we consider a mention-candidate pair as an instance, $(m_1, c_1), (m_1, c_2), (m_1, c_3), ..., (m_1, c_k)$, $(m_2, c_1), (m_2, c_2), (m_2, c_3), ..., (m_2, c_k)$, $(m_n, c_1), (m_n, c_2), (m_n, c_3), ..., (m_n, c_k)$. If we consider the classification problem as binary

classification then each instance is independent and a positive prediction means that a mention is mapped to a specific API element. If none of the mention's APIs match, then we consider the mention to be spurious. In multi-instance classification, all of the instances that belong to a particular mention m form a bag. The number of bags is equal to the number of mentions, while the number of instances in each bag is equal to the number of API candidates of this particular mention. The classification task marks a bag as positive if it contains an instance (m, c_j) that maps to an API. We identify this to be the instance with the highest sum of the normalized features within the positive bag. On the other hand, if a bag is predicted as negative, it is a spurious match and all of its instances are negative. As an implementation of a multi-instance classifier, we use the multi-instance Support Vector Machines (mi-SVM)[36], with its default settings (i.e. kernel=linear and maximum iterations=5000).

4.2 | Results

We first present our results evaluating **RQ1** with mentions as units. The evaluation uses the Random Forest classifier and 10-fold cross validation and a training set consisting of the same type of API element (i.e. method, class, field or combined). Table 4 shows the results split across different types of API elements, classes (or interfaces), methods, and fields. For a combination of API elements our technique shows higher precision (79%), lower recall (62%), with F1 score of 69%. Considering the different types of API elements, our technique performs best on fields, followed by classes, with methods performing the poorest, with an F1-score of 58%.

API Elements	Precision	Recall	F1 score
Class	87%	74%	80%
Field	94%	78%	84%
Method	69%	50%	58%
Combined	79%	62%	69%

TABLE 4 Results using mentions as units.

API Elements	Bag Level			Instance Level		
	Precision	Recall	F1 Score	Precision	Recall	F1 Score
Class	73%	100%	84%	99%	73%	84%
Field	–	–	–	–	–	–
Method	83%	79%	79%	56%	48%	51%
Combined	79 %	29%	42%	60%	50%	54%

TABLE 5 Results of multi instance classification.

We also evaluate our technique using tutorials as units, where we use mentions of a specific API type from 12 of the 13 tutorials as a training set and use the final tutorial as the test set.

The results of this evaluation are shown in Table 3. Again, we present the classifier's output divided into different types of API elements and combined. We observe high effectiveness on classes and fields, but much low values for method (average F1-score of 44%) or combined (average F1-score of 36%).

The low values on method across both of the evaluations motivate the formulation of the problem as multi-instance classification, which groups instances belonging to the same mention instead of treating them separately as in the previous formulations.

Using bags to represent each unique mention in the tutorial and instances to represent each mention - API element pair, the results for multi-instance classification are shown in Table 5.

For train test splits, we use the tutorials as units. The number of fields in the dataset was insufficient to produce results using this method so these results are omitted. Using this formulation of the problem we observe reasonable results at the bag level for methods and classes and at the instance level for classes. The per instance method results and the combined results were weaker

than class or field, as in the binary classification. However, compared to the binary classification we observe slight improvements in the results on methods, with F1-score of 51% ,and on the combination of all API elements, with an F1-score of 54%.

Tutorial Name	Version Range	Predicted Version Range	Manhattan Distance
Learning to Parse XML Data in Your Android App	[1-28]	[26-28]	25
Navigation Drawer Android Example	[22-27]	[25-27]	3
How to Get all Registered Email Accounts in Android	[5-22]	[5-22]	0
Scheduling Background Tasks in Android	[3-28]	[1-21]	9
Android Lollipop Swipe to Refresh Example	[22-28]	[24-28]	2
Android Navigation Drawer - for Sliding Menu / Sidebar	[25-28]	[24-28]	1
Building Android applications with Gradle - Tutorial	[1-28]	[1-28]	0
Android Facebook Login Tutorial - Integrating Facebook SDK 4	[26-28]	[25-27]	2
Using ViewPager to Create a Sliding Screen UI in Android	[25-28]	[25-26]	2
Retrofit, a Simple HTTP Client for Android and Java	[25-28]	[26-28]	1
Convert Speech to Text in Android Application	[26-28]	[25-28]	1
Android Chat Bubble Layout - with 9 patch Image using ListView	[25-28]	[26-28]	1
Understanding Androids Parcelable - Tutorial	[1-28]	[1-28]	0
Average Distance			3.61

TABLE 6 Comparison of the true and predicted version ranges of tutorials.

Based on these results, we cannot answer **RQ1** strongly in the affirmative for all API elements. While the results are sufficiently strong for fields and classes, the results for methods and for a combined mix of all API elements still miss a large set of mentions.

Next, for **RQ2**, we examine how the technique performs in determining the version ranges of the tutorials in our experimental set. While **RQ1** is a prerequisite for **RQ2**, we observe a high redundancy in mention types in a tutorial, which makes it possible to have reasonable results on **RQ2** even with subpar results on **RQ1**. For instance, `startActivityForResult` is mentioned numerous times in the tutorial listed in Figure 1.

The results for **RQ2** are shown in Table 6. Our method predicts the correct version ranges for three of the thirteen tutorials. For the ones whose versions are incorrectly predicted, the majority, ten out of twelve, are missed with very small margins of 1 or 2 versions. For two of the tutorials the predicted versions are significantly distant from the true ranges. This shows that even though some individual API mentions and their corresponding API-levels are incorrectly predicted by our technique, due to redundancy of mentions inherent in the tutorials, the upper and lower bound of the predicted version ranges are quite similar to the corresponding true version ranges. Tiny variances in the predicted version ranges are unlikely to significantly hinder the use of our technique.

Error Analysis. We qualitatively examined the results of our technique, focusing specifically on instances where our technique performed poorly. The tutorial Learning to Parse XML Data in Your Android App is one where we misclassify the version by a large margin, predicting 26 where the true minimum version of the APIs mentioned in this tutorial is 1. Examining all the potential mentions of this tutorial we observe that in the following textual segment of the tutorial, the word **write** is considered a potential mention.

> [...] With the help of these APIs you can easily incorporate XML into your Android app. The XML parsers do the tedious work of parsing the XML and we have to just **write** the code to fetch the appropriate information from it and store it for further processing. [...]

In this case, the mention to **write** clearly does not refer to an API, however, it is misclassified as the call the Android `write` API method:

```
void AsynchronousSocketChannel.write(ByteBuffer src, A attachment,
                                CompletionHandler<Integer, ? super A> handler)
```

This occurs because there are tokens in the method's API description text that match tokens in the context of this mention in the tutorial, providing a positive value for the *Text Similarity* feature. With this value for this feature, and the remaining features as zero, the classifier produces a false positive. As the **write** method is introduced in Android API 26, this results in a large error for this tutorial. Mitigating errors like the one described here likely requires introducing additional features as improvements to the classifier seem unlikely to be helpful, since many true positive mentions have the same feature values. This specific error persisted for both of the classifiers we applied in this paper. One additional feature that can be explored to improve this error is using word embeddings computed on a large external corpus, e.g. on Stack Overflow, to enrich the text of the API documentation with additional terms and improve the quality of the matching. Too often, we found that the Android API descriptions were very brief which made resulted in zeros for many of the features, as was the case here.

5 | RELATED WORK

To our knowledge, there is no prior research that focuses specifically on the validity of tutorials with respect to versions of APIs they reference. However, more broadly, we can divide the related research into two categories: 1) studies of API deprecation and versioning of APIs referenced in various contexts; and 2) techniques for extracting API mentions from different types of formal and informal documentation. We discuss each of these categories in turn.

Many studies of API deprecation focus on source code. Leveraging data in software repositories, a recent tool called APIDIFF detects the API changes between two versions of Java libraries [37]. APIDIFF serves as a warning system for client applications that rely on these libraries. The tool extracts syntactic changes in the evolution of a software repository, reporting a set of predefined breaking and non-breaking changes in API types, methods, and fields. Targeting Android application binaries distributed via the app store, the MAD-API framework detects API misuses, using a reverse engineering toolchain [10]. MAD-API detects API misuses based on a gold set of Android APIs, including per-version removals, deprecations, and additions. The APIs in the binary that are misused are detected and reported. Both of these tools focus on detecting API versioning problems in source code, a very different domain from versioning software development tutorials that consist of a combination of source code and natural language text.

The recently proposed framework DEPRECATION WATCHER [9] detects deprecated APIs in the source code snippets of StackOverflow posts. Since StackOverflow posts are concise and focus on a single topic [38,9], a lightweight tool like DEPRECATION WATCHER can be effective in detecting deprecated API while disregarding the natural language context of the post. However, researchers have pointed out that the content of the surrounding text is especially important in high quality answers on StackOverflow [38]. Our technique focuses on API versioning of tutorials, which are significantly more complex, often weaving together multiple related topics. The code segments in tutorials are also much more complex, which makes the regular expression based technique in DEPRECATION WATCHER potentially inadequate. However, more importantly, the natural language context in tutorials is much larger than StackOverflow posts and ignoring it is likely to lead to poor results.

Detecting API mentions in informal documentation is an active area of research, as, once detected, the API mentions can be used to improve documentation lookup efficiency or serve as the basis for a variety of recommendation systems. For instance, Ye et al. proposed a technique for detecting API mentions, despite difficulties introduced by polysemy and sentence-format variations in the informal documentation [39]. Ye et al. also utilized a conditional random fields classifier to detect the fully qualified names of API mentions, obtaining high precision and recall [40]. However, the chosen approach relies heavily on hints that are specific to the Q&A conversations in StackOverflow, such as the question title, StackOverflow tags (used to eliminate non-relevant candidates), as well as tags embedded in HTML to find out types (class and interface). These techniques are too specific to StackOverflow and therefore are not completely applicable to our problem.

A recently proposed tool also targeting StackOverflow mentions, named ANACE [8], leverages a number of generic features that could be applied to any informal documentation source. We modeled our features on some of the features shown to be effective by ANACE, however, our workflow is significantly simpler relative to this tool. Finally, we also differ from ANACE in targeting versioning of tutorials, rather than just API mention detection in StackOverflow posts.

Techniques for extracting API mentions from informal documentation are similar to the main technique we use for tutorial versioning. However, many of them extract API mentions for the purpose of generating documentation, so they do not assume access to the official API documentation as we do. RECODOC is a tool proposed to extract code-like terms in documents that describe API elements. The tool is based on a number of filtering heuristics[41], and relies on a parser to identify code-like terms, which can miss terms with formatting inconsistencies and result in false positives on similar terms (e.g URLs). ACE is a tool that uses local context extracted from a single document and global context extracted from the corpus to discover code-like terms in informal documentation[42]. The tool relies on tags to filter out those posts that do not represent APIs of interest and introduce noise[8]. ACE also uses an island parser and relies on a set of regular expressions which can be language or source specific. RECODOC was later applied in finding relevant tutorial segments for a given API element[43,44]. Both of these approaches only consider class and interface level granularity, but not method or field.

An unsupervised approach FRAPT[45] has recently been proposed to recommend relevant tutorial fragments of APIs. FRAPT relies on HTML tags and special keywords to identify API names, which can be unreliable for APIs that use common names. FRAPT does not aim to resolve the ambiguity in common API names. FRAPT[45] was later used to implement a framework named SOTU[46] that aims to find answers to API related natural language questions by utilizing fragments of API tutorials and StackOverflow.

Zhang et al. devised a technique that links API official documentation to the API related questions in StackOverflow, which utilizes the lexical similarity between StackOverflow questions and API description and the history of prior answered questions on StackOverflow[47]. An API recommendation technique called RACK[48] has been proposed to discover appropriate APIs for a given natural language query by utilizing crowdsourced knowledge mined from StackOverflow in the form of keyword-api association. While utilizing API mention discovery in their workflow, both of these techniques target Stack Overflow and aim at completely different software engineering problems than the tutorial versioning problem that is the focus of this paper.

6 | CONCLUSION AND FUTURE WORK

In this paper we propose a novel idea for inferring the version compatibility of informal software documentation, focusing specifically on written tutorials available on the Web. Many tutorials are read by numerous developers, especially by novices, and a system that can warn developers of version incompatibilities is likely to improve productivity by reducing the time spent struggling to learn a difficult development-related concept.

We perform a motivational study producing a manually-annotated corpus of 13 Android tutorials, obtained using stratified sampling that considers the source and year of publishing. Our study finds several tutorials with limited Android API version compatibility, including both dated tutorials that are not compatible with recent API releases and newer tutorials that are incompatible with older APIs.

We then focus on a developing a workflow that automates the task of versioning tutorials given (versioned) official API documentation. We decompose this task into two subtasks, one of determining whether a term that matches an API element (class, field, or method) is an actual API mention and the second of disambiguating a overloaded API name to the specific element it refers to. For these two tasks, we express a set of features and experimentally study different classification algorithms and problem setups to understand their effect on this problem. We find that classes and fields are straightforward to disambiguate, but that common API method names can be very challenging. However, we also find that tutorials possess sufficient redundancy in their API mentions and even with imperfect per-mention classification, the overall tutorial version can often be accurately recognized. Therefore, we observed that our approach is effective at determining the final valid version ranges of many of the Android tutorials we examined.

In the future, we plan to extend our research by exploring additional features that reflect the popularity of a particular API, as developers often mention popular APIs without much additional context. We also plan to experiment on larger and more diverse collections of tutorials. We also plan on exploring how version hints can be integrated into development environments and how developers perceive such suggestions.

7 | ACKNOWLEDGMENT

The authors acknowledge support for this work from the US National Science Foundation, under award number 1812968.

References

1. Brandt J, Guo PJ, Lewenstein J, Dontcheva M, Klemmer SR. Two Studies of Opportunistic Programming: Interleaving Web Foraging, Learning, and Writing Code. In: CHI '09. ACM; 2009; New York, NY, USA: 1589–1598

2. Chatterjee P, Nishi MA, Damevski K, Augustine V, Pollock L, Kraft NA. What information about code snippets is available in different software-related documents? an exploratory study. In: SANER 2017. IEEE; 2017; Washington, DC, USA: 382–386

3. Bao L, Xing Z, Wang X, Zhou B. Tracking and Analyzing Cross-Cutting Activities in Developers' Daily Work (N). In: ASE '15. IEEE Computer Society; 2015; Washington, DC, USA: 277–282

4. Xia X, Bao L, Lo D, Kochhar PS, Hassan AE, Xing Z. What Do Developers Search for on the Web?. *Empirical Softw. Engg.* 2017; 22(6): 3149–3185. doi: 10.1007/s10664-017-9514-4

5. Ponzanelli L, Bavota G, Mocci A, et al. Too Long; Didn'T Watch!: Extracting Relevant Fragments from Software Development Video Tutorials. In: ICSE '16. ACM; 2016; New York, NY, USA: 261–272

6. McDonnell T, Ray B, Kim M. An empirical study of api stability and adoption in the android ecosystem. In: ICSM '13. IEEE; 2013; Washington, DC, USA: 70–79

7. Ko D, Ma K, Park S, Kim S, Kim D, Le Traon Y. API document quality for resolving deprecated APIs. In: APSEC '14. IEEE; 2014; Piscataway, NJ, USA: 27–30

8. Uddin G, Robillard MP. Resolving API Mentions in Informal Documents. *CoRR* 2017; abs/1709.02396.

9. Zhou J, Walker RJ. API Deprecation: A Retrospective Analysis and Detection Method for Code Examples on the Web. In: FSE 2016. ACM; 2016; New York, NY, USA: 266–277

10. Luo T, Wu J, Yang M, Zhao S, Wu Y, Wang Y. MAD-API: Detection, Correction and Explanation of API Misuses in Distributed Android Applications. In: Springer International Publishing; 2018; Cham: 123–140

11. Tiarks R, Maalej W. How Does a Typical Tutorial for Mobile Development Look Like?. In: MSR 2014. ACM; 2014; New York, NY, USA: 272–281

12. Ko AJ, DeLine R, Venolia G. Information Needs in Collocated Software Development Teams. In: ICSE '07. IEEE Computer Society; 2007; Washington, DC, USA: 344–353

13. Singer J, Lethbridge T, Vinson N, Anquetil N. An Examination of Software Engineering Work Practices. In: CASCON '10. IBM Corp.; 2010; Riverton, NJ, USA: 174–188

14. Hora A, Robbes R, Anquetil N, Etien A, Ducasse S, Valente MT. How do developers react to API evolution? The Pharo ecosystem case. In: ICSME '15. IEEE; 2015; Washington, DC, USA: 251-260

15. Espinha T, Zaidman A, Gross HG. Web API growing pains: Stories from client developers and their code. In: CSMR-WCRE '14. IEEE; 2014; Washington, DC, USA: 84–93

16. Sawant AA, Robbes R, Bacchelli A. On the reaction to deprecation of 25,357 clients of 4+ 1 popular Java APIs. In: ICSME '16. IEEE; 2016; Washington, DC, USA: 400-410

17. Hou D, Yao X. Exploring the Intent behind API Evolution: A Case Study. In: . 00. ; 2011: 131-140

18. Venkatesh PK, Wang S, Zhang F, Zou Y, Hassan AE. What Do Client Developers Concern When Using Web APIs? An Empirical Study on Developer Forums and Stack Overflow. In: ICWS' 16. IEEE Computer Society; 2016; Washington, DC, USA: 131-138

19. Linares-Vásquez M, Bavota G, Di Penta M, Oliveto R, Poshyvanyk D. How Do API Changes Trigger Stack Overflow Discussions? A Study on the Android SDK. In: ICPC 2014. ACM; 2014; New York, NY, USA: 83–94

20. Kavaler D, Filkov V. Determinants of quality, latency, and amount of Stack Overflow answers about recent Android APIs. In: . 13. Public Library of Science; 2018; California, US: e0194139

21. Android StartActivityForResult Example. https://www.javatpoint.com/android-startactivityforresult-example; 2018.

22. MCREYNOLDS J. Android Tutorial: Implement A Shake Listener. http://jasonmcreynolds.com/?p=388; 2013.

23. Android Official Documentation. https://developer.android.com/reference/; 2018.

24. Android API Differences Report. https://developer.android.com/sdk/api_diff/../changes; 2018.

25. Android API Differences Report For Support Library. https://developer.android.com/sdk/support_api_diff/.../changes; 2018.

26. Android Version History. https://en.wikipedia.org/wiki/Android_version_history; 2018.

27. NLTK 3.3 Documentation. https://www.nltk.org/; 2018.

28. TextBlob: Simplified Text Processing. https://textblob.readthedocs.io/en/dev/; 2018.

29. SrcML Tool Documentation. https://www.srcml.org/; 2018.

30. Sklearn Metrics Precision Recall Fscore Support. http://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html; 2018.

31. Accuracy, Precision, Recall or F1?. https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9; 2018.

32. Sklearn Metrics Precision Recall F1score. http://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html/; 2018.

33. Taxicab Geometry. https://en.wikipedia.org/wiki/Taxicab_geometry; 2018.

34. Random Forest Classifier. http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html; 2018.

35. Multi Instance Classifier. https://github.com/garydoranjr/misvm; 2018.

36. Andrews S, Hofmann T, Tsochantaridis I. Multiple Instance Learning with Generalized Support Vector Machines. In: American Association for Artificial Intelligence; 2002; Menlo Park, CA, USA: 943–944.

37. Brito A, Xavier L, Hora A, Valente MT. APIDiff: Detecting API breaking changes. In: SANER '18. IEEE Computer Society; 2018; Washington, DC, USA: 507-511

38. Nasehi S, Sillito J, Maurer F, Burns C. What makes a good code example?: A study of programming Q&A in StackOverflow. In: ICSM 2012. IEEE; 2012; Washington, DC, USA: 25–34

39. Ye D, Xing Z, Foo CY, Li J, Kapre N. Learning to extract api mentions from informal natural language discussions. In: ICSME 2016. IEEE; 2016; Washington DC, USA: 389–399

40. Ye D, Bao L, Xing Z, Lin SW. APIReal: an API recognition and linking approach for online developer forums. In: Springer; 2018; New York, NY, USA: 1–32

41. Dagenais B, Robillard MP. Recovering Traceability Links Between an API and Its Learning Resources. In: ICSE '12. IEEE Press; 2012; Piscataway, NJ, USA: 47–57

42. Rigby PC, Robillard MP. Discovering Essential Code Elements in Informal Documentation. In: ICSE '13. IEEE Press; 2013; Piscataway, NJ, USA: 832–841

43. Petrosyan G, Robillard MP, De Mori R. Discovering Information Explaining API Types Using Text Classification. In: ICSE '15. IEEE Press; 2015; Piscataway, NJ, USA: 869–879

44. Rigby PC, Robillard MP. A more accurate model for finding tutorial segments explaining APIs. In: SANER '16. IEEE Press; 2016; Piscataway, NJ, USA: 157-167

45. Jiang H, Zhang J, Ren Z, Zhang T. An Unsupervised Approach for Discovering Relevant Tutorial Fragments for APIs. In: ICSE '17. IEEE Press; 2017; Piscataway, NJ, USA: 38–48

46. Wu D, Jing XY, Chen H, et al. Automatically Answering API-related Questions. In: ICSE '18. ACM; 2018; New York, NY, USA: 270–271

47. Zhang J, Jiang H, Ren Z, Chen X. Recommending APIs for API Related Questions in Stack Overflow. In: IEEE '13. IEEE; 2013; Piscataway, NJ, USA: 6205-6219

48. Rahman MM, Roy CK, Lo D. Rack: Automatic api recommendation using crowdsourced knowledge. In: SANER '16. IEEE Press; 2016; Piscataway, NJ, USA: 349-359