2019

ASSESSING THE QUALITY OF SOFTWARE DEVELOPMENT TUTORIALS AVAILABLE ON THE WEB

Manziba A. Nishi

ASSESSING THE QUALITY OF SOFTWARE DEVELOPMENT TUTORIALS AVAILABLE ON THE WEB

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University.

by

MANZIBA AKANDA NISHI

Student degree with university - Dates

Director: Kostadin Damevski,

Assistant Professor, Department of Computer Science

Virginia Commonwealth University

Richmond, Virginia

December, 2019

Acknowledgements

First, I would like to acknowledge and express my sincere gratitude to my honorable advisor Dr. Kostadin Damevski, for his direction, assistance, and guidance. I would like to thank Dr. Preetam Ghosh, Dr. Bridget Thomson-McInnes, Dr. Eyuphan Bulut, Dr. Nicholas A. Kraft and Dr. Hui Chen for their kind approval to join my dissertation committee and for providing guidance and valuable feedback. I would like to thank my family members especially my husband and my mother, for their continuous support, encouragement and sacrifices they have made for me in my Ph.D journey. I would also like to thank my cute little son for being extremely patient while I was working on my dissertation.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Abstract

ASSESSING THE QUALITY OF SOFTWARE DEVELOPMENT TUTORIALS AVAILABLE ON THE WEB

By Manziba Akanda Nishi

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at Virginia Commonwealth University.

Virginia Commonwealth University, 2019.

Director: Kostadin Damevski,
Assistant Professor, Department of Computer Science

Both expert and novice software developers frequently access software development resources available on the Web in order to lookup or learn new APIs, tools and techniques. Software quality is affected negatively when developers fail to find high-quality information relevant to their problem. While there is a substantial amount of freely available resources that can be accessed online, some of the available resources contain information that suffers from error proneness, copyright infringement, security concerns, and incompatible versions. Use of such toxic information can have a strong negative effect on developer's efficacy. This dissertation focuses specifically on software tutorials, aiming to automatically evaluate the quality of such documents available on the Web. In order to achieve this goal, we present two contributions: 1) scalable detection of duplicated code snippets; 2) automatic identification of valid version ranges.

Software tutorials consist of a combination of source code snippets and natural

language text. The code snippets in a tutorial can originate from different sources, perhaps carrying stringent licensing requirements or known security vulnerabilities. Developers, typically unaware of this, can reuse these code snippets in their project. First, in this thesis, we present our work on a Web-scale code clone search technique that is able to detect duplicate code snippets between large scale document and source code corpora in order to trace toxic code snippets.

As software libraries and APIs evolve over time, existing software development tutorials can become outdated. It is difficult for software developers and especially novices to determine the expected version of the software implicit in a specific tutorial in order to decide whether the tutorial is applicable to their software development environment. To overcome this challenge, in this thesis we present a novel technique for automatic identification of the valid version range of software development tutorials on the Web.

CHAPTER 1

INTRODUCTION

During software development, developers often rely on external documentation, such as tutorials, blogs, and Q&A forums, to learn information on software artifacts, frameworks, libraries, and services [1, 2, 3]. However, research studies have shown that sometimes these resources fail to provide high quality information crucial to producing reliable software efficiently. While maintaining a high quality information corpus in popular Q&A sites, such as Stack Overflow, is challenging [4], consistent quality in other software development resources, such as software development tutorials, development blogs, etc., is even more difficult as such resources do not have a community-driven quality control system.

Among these software development resources, online tutorials are a particularly valuable source of community created information. For many popular technologies, tutorials present the definitive source of information on how to complete a task; they are often the means by which a vendor describes how their API should be used for a specific purpose. Tutorials provide information in an incremental fashion, using alternating segments of code, natural language descriptions, or diagrams, which can teach developers how to perform a specific task or a new skill more systematically than e.g. Q&A posts. Experienced software developers can share their procedural or how to knowledge in the form of software development tutorials.

There is no standard set of rules on how tutorials should be composed, so each varies based on the author's preferences. Tutorials can also be presented in different mediums, such as written documents, interactive programs, or screen recordings

(audio or video) [5]. Assessing the quality of software development resources is important as developers find it difficult and time consuming to distinguish among a huge number of similar documents, and, therefore, developers typically rely on simple cognitive heuristics such as the location of content and how much screen space is occupied [6]. The growing amount of low-quality content available online, e.g., in Q&A communities like Stack Overflow, can be potentially attributed to specific groups of contributors (e.g help vampires, noobs and reputation collectors) [7]. Automatically evaluating and recommending high quality software development resources for specific developers is a challenging task as the documents are complex and their quality can be decomposed across several dimensions, e.g. version incompatibility, license violations, security vulnerabilities etc.

Once written, tutorials are rarely actively updated (or removed) and can become dated over time. Some tutorials reference APIs that change rapidly, and deprecated classes, methods and fields can render tutorials inapplicable to newer releases of the APIs. Newer tutorials may not be compatible with older APIs that are still in use. Some tutorials direct developers towards poorly written or poorly maintained APIs. Developers often assume perfection behind APIs, rarely taking the time to evaluate the quality of APIs their software relies on. Use of highly unstable and rapidly changing APIs can be threat to the success and security of implemented apps or software, and as a result, bugs in these unstable APIs can drastically impact the source code quality [8]. Therefore, it is important that software development tutorials reflect the most up to date and high quality APIs for a specific purpose.

Source code is an integral part of software development tutorials. One research study reports that 70% of software development tutorials contain source code examples [5]. Developers often reuse (i.e., copy and paste) code available on tutorials into their projects. Changes in APIs can also make the example source code snippets

present in tutorials outdated, perhaps containing API misuses that lead to program crashes, resource leaks, and incomplete actions [9]. Some tutorials are poorly written to begin with. Buggy, unreliable, harmful code snippets present in tutorials can have an adverse downstream effect on software systems [10, 11, 12, 9]. Out of 1.3 million analyzed Android applications, 15.4% contain security related code snippets reused from Stack Overflow. Alarmingly, 97.9% of these applications contain at least one insecure code snippet [12].

License violations are another negative side effect of poor tutorials. Analyses conducted by researchers indicate developers are often unaware of the license of code snippets avialable on the Web [13] [14]. For instance, a large portion of Stack Overflow answerers (69%) and visitors (66%) never checked for license conflicts. Further, 138 out of 201 highly reputed participants (answerers) of a survey are unaware of checking licensing conflicts between code snippets reused or copied in their software projects and Stack Overflow's CC BY-SA 3.0 license. More alarmingly, 73 out of 87 Stack Overflow visitors are not even aware of its CC BY-SA 3.0 license [10].

This dissertation contributes to the state of the art in assessing the quality of the software development tutorials. While many researchers have contributed to evaluation of the quality of content on Q&A websites such as Stack Overflow and invented several quality measures, limited research thus far has focused on the evaluation of the quality of software development tutorials [5].

1.1 Motivation

Evaluating the quality of the software development tutorials is a potentially impactful area of research. However, quality can be expressed in different ways depending of the type of software and its domain as well as on individual perspective and interest [18]. Our perspective is that the quality of the software development tutorial

Fig. 1.: Portion of an Android tutorial utilizing the deprecated API `android.util.FloatMath` [15].

depends on the API mentions, code snippets and corresponding information present in them. Presence of outdated APIs and code snippets with known security problems and license restrictions degrade the overall quality of tutorials. In other words, the inappropriateness of the APIs mentioned in the tutorial can render the coding example and the corresponding information of the software development tutorials obsolete or even toxic.

According to a recent survey of developer, the second most popular suggestion made by software developers towards content quality on Stack Overflow is to intro-

https://stackoverflow.com/questions/15577307/how-to-extend-sqlitedatabase-class/15577409#1557

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
  Log.w(MySQLiteHelper.class.getName(),
      "Upgrading database from version " + oldVersion + " to "
          + newVersion + ", which will destroy all old data");
  db.execSQL("DROP TABLE IF EXISTS " + TABLE_COMMENTS);
  onCreate(db);
}
```

https://www.vogella.com/tutorials/AndroidSQLite/article.html

```
@Override
public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    Log.w(MySQLiteHelper.class.getName(),
            "Upgrading database from version " + oldVersion + " to "
                    + newVersion + ", which will destroy all old data");
    db.execSQL("DROP TABLE IF EXISTS " + TABLE_COMMENTS);
    onCreate(db);
}
```

Fig. 2.: Code snippet copied between a tutorial (lower portion) and a Stack Overflow (upper portion) post with different licensing terms [16], [17].

duce a mechanism to detect outdated content. For instance, one developer suggested: *"Have explicit mechanisms for dealing with content that goes out of date due to platform or language changes"* [19]. Some other developers suggested a traceability link between each code snippet and the API version under which it will work, stating: *"Clear associates between the codes snippets the versions of the API under which it will work. This is particularly when working with APIs that change frequently, like iOS and Unity"*. Furthermore, developers also wanted outdated answers to be deprecated and removed *"Make date important in marking outdated code, and deprecate those snippets via the community"* [19].

While prior research exists on aspects of quality of posts in Stack Overflow, far fewer approaches have focused on software development tutorials [19], which present many notable differences to Stack Overflow, the most notable of which is the lack

of community driven quality control (via up/down votes, accepted answers, etc.). As the same time, this software documentation source is very widely used, as many development tasks are difficult without the help of an appropriate tutorial.

As a motivating example, a portion of an Android tutorial is shown in Figure 9, where the tutorial shows an example code snippet that utilizes the `FloatMath` class. This class was deprecated in Android API level 22 and removed in level 23. At the time of writing, the most recent level of Android API is 28, showing clearly a large gap in the applicability of the code snippet in the tutorial. While this class is not the focal point of this tutorial, reusing the code snippet that references it can lead to difficulty and wasted 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 the tutorial's age is not so easy to discern.

A complimentary motivating example focusing on code snippet licensing is shown in Figure 2, showing a duplicate code snippet appearing in both a tutorial and Stack Overflow. In either of these sources there is no reference to the other, from which it could be understood where the tutorial originates from. Stack Overflow is available using the terms of one license: Creative Commons Attribute - ShareAlike 3.0 Unported (CC BY-SA 3.0). The specific tutorial licenses its code snippets according to the Eclipse Public License 2.0. While both licenses are fairly unrestrictive, and impose only minor constraints on reuse, a developer reusing this code has to interpret which license takes precedence. Many other tutorial sources impose much more restrictive terms on code reuse, which could lead to even larger problems when code is reused.

The two highlighted examples are just ones of many, while other similar problems, such as known security violations disseminated via tutorials, also exists. Therefore,

in this dissertation, as a broad goal, we target improving the quality of tutorials. The techniques we describe address this goal in part, focusing mainly on the perspective of a developer who is reading the tutorial, who we aim to inform of the quality concerns with the source. An alternative approach is to inform the tutorial writer, but, we have observed that in many cases the writer is difficult to reach while, at times, the tutorial can still be valid for a small audience that e.g., still uses a very old version of an API.

1.2 Contribution of the Research

The contributions of this dissertation to the problem of assessing the quality of software tutorials are the following:

- Web-Scale Textual Code Clone Search.

  We focus on improving the scalability of Web-scale textual code clone search, relative to current state of the art techniques. We target searching and extracting similar or near similar source code snippets between known quality sources (e.g. official API documentation is a high quality source, while a known software vulnerability database has code snippets of low quality) and unknown quality sources like tutorials. Due to the volume of the information available on the Web, it is important to detect source code snippets with reasonable speed and scalability. We propose a novel approach *Web-Scale Textual Code Clone Search* based on the *adaptive prefix filtering technique*. Our proposed approach improves the performance of code clone search and code clone detection, for many common execution parameters, when tested on common benchmarks. The experimental results exhibit improvements for commonly used similarity thresholds of between 40% and 80%, in the best case decreasing the execution

time up to 11% and increasing the number of filtered candidates up to 63%.

- Characterizing Duplicate Code Snippets between Stack Overflow and Tutorials.

  Developers are usually unaware of the quality and lineage of information available on popular Web resources, leading to potential maintenance problems and license violations when reusing code snippets from these resources. In this dissertation, we study the duplication of code snippets between two popular sources of software development information: the Stack Overflow Q&A site and software development tutorials. Our goals are to empirically understand the scale of repeated information between these two sources, to gain insight into why developers copy information from one source to the other, and to understand the evolution of duplicated information over time. To this end, we correlate a set of nearly 600 tutorials on Android available on the Web to the SOTorrent dataset, which isolates code snippets from Stack Overflow posts and tracks their changes over time. Our findings reveal that there are over 1,400 duplicate code snippets related to Android on Stack Overflow. Code that was duplicated on the two sources is effective at answering Stack Overflow questions; a significant number (31%) of answers that contained a duplicate code block were chosen as the accepted answer. Qualitative analysis reveals that developers commonly use Stack Overflow to ask clarifying questions about code they reused from tutorials, and copy code snippets from tutorials to provide answers to questions.

- Automatic Identification of Valid Version Range of Software Development Tutorials on the Web.

  To detect both updated and outdated information of the software development resources it is important to evaluate the version or range of versions of the information present in it. The high-level information cloud contains most relevant,

authentic as well as updated information. At the same time high-level information cloud stores previous versions of the information that helps us to evaluate the versions of information present in the low-level software development resources such as software tutorials. We propose a novel approach of automatic identification of valid version range of software development tutorials on the Web.

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.

1.3 Organization of the Dissertation

Chapter 2 presents the background and related work of our research. We discuss existing and prevalent methodology, techniques, research studies and surveys concerning the information quality of popular software development resources, scalable

code clone search and detection techniques, and versioning of software development resources.

Chapter 3 presents a detailed explanation about our proposed Web-scale textual code clone search technique, discussing types of code clones, problem formulation of code clone search and detection, related properties and lemmas. We have also presented experimental results compared to existing techniques. We also present the characterization and properties of the duplicate code snippets in between Stack Overflow and software tutorials. Using qualitative analysis we examined these code clone pairs to understand why developers usually copy code snippets between Stack Overflow and software tutorials. We have also presented several research findings based on the evolution of copied code snippets.

Chapter 4 presents the detailed methodology and result of an empirically study of the tutorial versioning problem. We discuss the workflow of our proposed approach of automatic identification of range of versions of tutorials, various machine learning features we have utilized, corresponding machine learning algorithms we have applied and a discussion of the experimental results.

Chapter 5 presents our future research plan. Finally, Chapter 6 presents the conclusions of our dissertation.

CHAPTER 2

BACKGROUND AND RELATED WORK

Numerous software development channels (e.g., developer blog posts, bug reports, mailing lists, code reviews, API documentation and Q&A forums) are available on the Web, enabling developers to communicate with each other regarding various programming language concepts, current state-of-art tools and technologies, programming problems, known bugs, etc. The body of software development resources available on the Web provides a great opportunity for developers to quickly learn or lookup answers to pressing software development problems. [20]. Online information is accessed multiple times daily by both seasoned and novice software developers. Effective search of the most relevant information online for a particular problem has become a relevant development skill. However, assessing the quality of that information is harder than finding it, while neglecting its quality can lead to lost productivity, technical debt or other downstream issues in the quality of the produced software.

2.1 Quality of Online Software Development Resources

Software development resources available on the Web can be deficient in a few ways: redundant content, outdated or incompatible information, content that is irrelevant or off topic, and information that may not be explained clearly or is mainly opinion-based. Such negative aspects of software documentation make readers impatient, confused, and uninterested to use these resources in the future [22].

Researchers have proposed several models and metrics to quantitatively and objectively evaluate the quality of these software development resources available on

the web. These models and metrics can measure the quality of the software development resources based on different dimensions such as trustworthiness, correctness, relevancy, comprehensiveness, and stability [18]. In addition, various features have been introduced to quantitatively assess the quality of Q&A websites such as Stack Overflow. Some of these features include textual or word-based representations of posts such as size of a question, size of an answer, similar words between a question and its answer, length ratio between a question and its answer. Quality is also measured based on the activity of the participants such as his/her communal interactions, reputations and popularity. Moreover, on Stack Overflow, the number of up votes, down votes, best answers, accepted answers, quality ratings as well as editor and user's reputations are useful attributes to measure post quality [23].

Software continuously evolves and this can make existing software development resources obsolete. Changes of API versions can make the existing source code snippets in software-related documents no longer applicable. Although questions about new APIs generally attract more attention, well thought out answers need more time to emerge and be recognized by the community [24].

Code snippets are an important factor for assessing the quality of software development resources. In Q&A sites, the questions elaborated with at least one code snippet usually get multiple good quality answers [25]. Answering problem specific questions is extremely difficult if no code snippet is included [26]. Developers often find it problematic to reuse the source code snippets on platforms such as Stack Overflow due to the lack of quality assessment [19]. The reuse of the outdated code snippets present in the software development resources can impose potential risk of security vulnerabilities [27], while developers face difficulty recognizing obsolete code snippets that are not appropriate to their specific configuration [19]. Moreover, developers are also at potential risk of violating license agreement when they simply

copy and paste the code snippets from one project to another without considering license agreements. Outdated code snippets, code snippets that are violating software licenses, and code snippets with security vulnerabilities are referred as toxic code snippets. The presence of such obsolete code snippets has a strong negative effect on the quality of online resources [27] [10] [28].

A recent survey finds that significant number (1,279) of code snippets copied from Stack Overflow to Android apps potentially violate the software license [10] [13]. Although Stack Overflow participants are usually notified of the outdated code snippets present, almost 19% of them never fix these code snippets [27]. The quality of content of Q&A website also degrades due to improper curation, e.g., when incorrect tags are assigned to the posts or if the topic of the contents is no longer applicable or highly subjective. A high quality question is an important factor for a high quality answer [25].

Researchers have also revealed that a significant number(29%) of security related code snippets of Stack Overflow have potential security breaches, while having been reused by over 1 million Android apps on Google play [12] [9] and out of 217,818 analyzed Stack Overflow posts, around 67,543 posts may have potential API usage violations that may impose the risk of producing unexpected output and behavior [9]. Recently researchers have evaluated the security susceptibility of the Stack Overflow code snippets in terms of social coding properties such as questions, answers, code snippets, different user attributes, badges as well as attributes related to code content such as APIs, function names, method names. The research study reveals different types of security risks are associated to defective code snippets. Among these, Android Manifest Configuration imposes the most dominating security breach (28.73%) when code snippets are copied from Stack Overflow. The Android apps are in danger of potential security concerns because insecure code snippets often request several

unnecessary permissions. The second most dominating security risk is related to data security (23.05%) which often leads to password sniffing attacks [29].

2.2 Code Clone Detection

The problem of finding problematic code snippets can be formulated as similarity-based search, where a code snippet with a known issue is compared to the corpus of code snippets in online resources. Code clone detection is a research area that focuses on finding similar code pairs. In order to match a problematic code snippet across a large corpus, e.g., Stack Overflow, only very scalable code clone detection algorithms suffice.

Scaling code clone detection to work across multiple repositories is a specific area of interest among researchers. Popular and notable examples of large-scale code clone detection include CCFinderX [36], which is one of the foremost token based code clone detection tools able to scale up to large repositories and detect Type-1 and Type-2 code clones. In [37], an inverted index-based approach was first proposed, detecting Type-1 and Type-2 clones. Deckard [38] is another tool that aims to scale to large source code repositories. It uses a tree-based data structure and detects clones by identifying similar subtrees and is able to detect up to Type-3 code clones. NiCad [39] is a scalable code clone detection tool that can detect Type-3 clones using a technique based on parsing, normalization and filtering. When compared using similar execution parameters, CCFinderX scales up to 100 million lines of code, Nicad scales up to 10 million lines of code, while Deckard scales up to 1 million lines of code [40].

Parallel, distributed or online (i.e. incremental) techniques add another dimension in examining scalable code clone detection technique. For instance, iClone [41] is the first incremental code clone detection technique that detects code clones in the current version of code repository by leveraging executions on previous versions of

the same repository. It uses a suffix tree-based and token-based approach that can detect Type-1 and Type-2 code clones. A scalable distributed code clone detection tool named D-CCFinder was proposed in [42], which can scale code clone detection in a distributed environment to detect Type-1 and Type-2 clones. In [43] a scalable code clone detection technique has been introduced where input files are partitioned into smaller subsets and a shuffling framework is utilized to allow the clone detection tool to execute on each of the subsets separately, enabling it to detect code clones in parallel.

Recently, the SourcererCC [40] code clone detection tool proposed a token-based prefix filtering code clone detection technique, which greatly reduces the number of candidate code clone pairs, enabling it to detect up to Type-3 clones in Internet-scale source code repositories. SourcererCC is the best scaling tool on a single machine that we are aware of. The approach described in our prposed technique extends SourcererCC with an adaptive approach that allows for even greater gains in performance for large-scale source code datasets.

SourcererCC is based on two filtering heuristics, prefix filtering and token position filtering, which reduce the number of candidate pairs that require costly pairwise comparison of all of their tokens [44, 45]. These two filtering heuristics attempt to rapidly, with few token comparisons, detect pairs of code blocks that diverge very significantly from each other. To perform this task, a subset (or prefix) is isolated in each of the two code blocks, where if there are no matching tokens in the subsets then we can safely reject them as a candidate pair, without proceeding further, and without attempting to compare all of their tokens. On the other hand, a single matching token in the subset allows the pair to proceed to further scrutiny as a code clone.

Recently, an extremely scalable code clone detection tool VUDDY [46] has been presented. VUDDY's purpose is to detect vulnerable code clones for security improve-

ment. While VUDDY has been shown to be significantly faster than SourcererCC it is designed to only detect Type-1 and Type-2 code clones. So far, the precision and recall of VUDDY has only been evaluated for relatively few instances of code with security vulnerabilities.

Wang et al. [47] recently proposed an additional filtering heuristic to those used in SourcererCC, called *adaptive prefix filtering*. This technique posits that deeper prefix lengths, which attempt more aggressive filtering at a higher performance cost, can achieve good performance on some types of input. An adaptive prefix filtering technique attempts to estimate the right level of filtering for each candidate by optimizing the trade-off between the cost of deeper filtering with the benefit of reducing the number of candidates. We have applied adaptive prefix filtering to code clone detection, and evaluates it's adequacy.

2.3 Code Clone Search

In this section we describe some of the previous approaches directed towards code clone search, which are significantly fewer than those that target code clone detection. Our technique targets both code clone detection and code clone search.

Multidimensional indexing structures have previously been proposed for code clone search. The technique proposed by Lee et al. [30] captures semantic information via a characteristic vector containing the occurrence counter of each syntactic element. This technique also retrieves a ranked list of code clones for each query, similar to typical information retrieval techniques. Another code clone search approach that uses multi-level indexing to detect Type-1,Type-2, Type-3 code clone was proposed by Kelvanloo et al. [31].

SeClone is a technique that targets code clone search [32] using a complex workflow, which includes the creation of Java ASTs for each file, two types of indices (code

pattern index and type usage index), a tailored search algorithm for code search, and a post-processing step that leverages clone pair clustering and grouping. Another complex technique Internet scale code clone search technique has been recently proposed, performing an abstracted code search for working code examples for reuse [33]. This technique uses an abstract representation of code snippets (using p-strings [34]), which are searched using frequent itemset mining. The result set of working code example is ranked using relevance of code patterns, popularity of abstract solutions, and similarity of code snippets to the output code snippets [33].

A clone search technique proposed by Koschke et al. [35] detects clones between two different systems. The technique uses a suffix tree to represent the code base that is smaller between the two. Subsequently, every file of the other system is compared against the suffix tree. A hash-based technique is additionally used for reducing the number of file comparisons. The technique only detects Type-1 and Type-2 code clones.

All of the above approaches towards code clone search utilize complex data structures to represent code blocks and (or) leverage complicated indexing schemes. Different from code clone detection, code clone search techniques typically do not optimize the index building time, since it is considered to be a fixed cost. Our proposed approach applies to both code clone search and code clone detection. Unlike other approaches for code clone search, it uses simple token-based code block representation that is fast and scalable to build and also maintains the ability to produce reasonable detection accuracy for Type-1, Type-2 and Type-3 code clones.

2.4 Valid Version Range of Software Development Resources

To our knowledge, there is no prior research that focuses specifically on the validity of tutorials with respect to versions of APIs they reference. 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 [48]. 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 [49]. 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 [50] detects deprecated APIs in the source code snippets of StackOverflow posts. Since StackOverflow posts are concise and focus on a single topic [51, 50], 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 [51]. Our technique focuses on API versioning of tutorials, which are significantly more complex, often weaving together multiple related topics. The code snippets in tutorials are also much more complex, which makes the regu-

lar 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 [52]. Ye et al. also utilized a conditional random fields classifier to detect the fully qualified names of API mentions, obtaining high precision and recall [53]. 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 [54], 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.

RECODOC is a tool proposed to extract code-like terms in documents linked to API elements. The tool is based on a number of filtering heuristics [55], 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 [56]. The tool relies on tags to filter out those posts that do not represent APIs of interest and introduce noise [54]. 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 [57, 58]. Both of these approaches only consider class and interface level granularity, but not method or field. To detect API versions of a tutorial, we need to consider all API element, including classes (or interfaces), fields and method. Therefore RECODOC is not directly applicable to the tutorial versioning problem.

An unsupervised approach FRAPT [59] 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, and does not do anything to resolve the ambiguity in common API names. FRAPT [59] was later used to implement a framework named SOTU [60] that aims to find answers to API related natural language questions by utilizing fragments of API tutorials and StackOverflow.

Zhang et al. devised a recommendation system that links API official documentation to the API related questions in StackOverflow [61]. They utilized the lexical similarity between StackOverflow questions and API description and the resolved history of prior answered questions on StackOverflow. An API recommendation technique called RACK [62] 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 recommendation techniques aim at completely different software engineering problems than the tutorial versioning problem that is the focus of our research work.

CHAPTER 3

WEB-SCALE TEXTUAL CODE CLONE SEARCH

Source code snippets in software development resources on the Web are important to the quality of these documents. Given a known toxic code snippet, it is challenging to use existing tools and techniques to find similar toxic source code snippets in online software development resources, due to the large volume of information in these sources. In this thesis, we propose a technique for *Web-scale textual code clone search* that can be used for retrieving similar source code snippets in software development resources on the Web. Most existing code clone search and detection techniques have difficulty scaling up to extremely large corpora of source code [43, 76].

Traditionally, developers introduce clones in a code base mainly when reusing existing code snippets without significant alteration, or when certain code snippets are implemented by developers following a common mental macro [63, 64, 36]. Researchers have shown that developers tend to perform software maintenance tasks more effectively when they are searching for code clones and have the results of code clone detection [65, 66]. Empirical studies have noted that code clones are widespread, and that a significant portion of source code (between 5% and 20%) is copied or modified from already implemented code snippets [63, 67, 68].

Performing code clone detection across numerous software repositories is a common use case. Specific applications for large scale code clone detection include querying library candidates [69], categorizing copyright infringement and license violations [70, 71], plagiarism detection [71, 70], finding product lines in reverse engineering [72, 71], tracing the origin of a component [73], searching for code snippets in

large software repositories [74, 31], and spotting analogous applications in Android markets [75, 40].

Similar or nearly similar source code snippets that encode the same algorithm are called code clones [40]. Code clone search is an area of research where a single code snippet is supplied as a query to be matched in large source code corpus, retrieving a list of code clones. Unlike conventional code clone detection, code clone search requires the source code to be pre-indexed and for the similarity threshold between the query block and it's clones to be specified at query time, instead of, at indexing time. This twist makes it challenging for typical code clone detection to be used without modification, and requires a more flexible data structure. During the development process, software developers often use code clone search to locate reusable similar or nearly similar source code snippets from other related projects [30].

Types of Code Clones. A continuous portion of source code is referred to as code snippet [40]. Based on the nature of the similarity between code snippets, the software engineering community has identified four types of code clones, by which code clone detection techniques can be organized.

- Type-1 (T1): Syntactically equivalent code snippets are called Type-1 clones. These code snippets may have differences in whitespace, layout and comments.
- Type-2 (T2): Code snippets that are syntactically comparable but are slightly contrasting in terms of variable names, function names, or identifier names.
- Type-3 (T3): If two code snippets contain statements that have been inserted, altered, expunged, there is a gap in statements, or statement order differs, then these are called Type-3 clones.
- Type-4 (T4): Semantically equivalent code snippets are called Type-4 clones.

As an example, Table 1 shows Type-1, Type-2, Type-3 and Type-4 code clone of the original code snippet [80].

//Original Code Snippet	//Type-1 Code Clone	//Type-2 Code Clone	//Type-3 Code Clone	//Type-4 Code Clone

`if (a==b)`
`{`
`c=a*b; //C1`
`}`
`else`
`c=a/b; //C2`
	`if (a==b)`
	`{`
	`//comment1`
	`c=a*b;`
	`}`
	`else`
	`//comment2`
	`c=a/b;`
		`if (g==f)`
		`{//comment1`
		`h=g*f;`
		`}`
		`else`
		`//comment2`
		`h=g/f;`
			`if (a==b)`
			`{//comment1`
			`c=a*b;`
			`// New Stat.`
			`b=a-c;`
			`}`
			`else`
			`//comment2`
			`c=a/b;`
				`switch(true)`
				`{//comment1`
				`case a==b:`
				`c=a*b;`
				`//comment2`
				`case a!=b:`
				`c=a/b;}`

Table 1.: Types of code clones [80].

3.1 Contribution

Among the tools aimed towards large scale code clone search and detection, a common limitation is in the complexity of differences among clones they can detect. For instance, scalable token based approaches [36, 37, 69] have difficulty detecting near miss (Type-3) code clones, which can occur more frequently than other types of clones [76, 77, 78]. Parallel and distributed clone detection techniques like D-CCFinder [42] can be more burdensome to manage, requiring specialized hardware or software support, while tree based code clone detection technique, such as Deckard [38], place higher demands on memory. The restrictions of these existing tools and techniques motivated us to propose a *Web-scale textual code clone search technique* that can work in very large source code repository and other software development resources on the Web, with high speed and low memory cost.

We propose a token-based code clone search technique aimed at scalability and detecting Type-3 clones, consisting of two main steps: filtering and verification. In the filtering step we aim to significantly reduce the number of code snippets for comparison, removing from consideration snippets that do not have any possibility of

being code clones. In the verification step we determine whether candidate pairs that survived the filtering step are really code clones. This two step process greatly reduces the runtime of code clone search and code clone detection, allowing the technique to scale up to very large corpora. This technique is based on the *adaptive prefix filtering heuristic* [47], which is an extended version of *prefix filtering heuristic* [44] [45] previously applied towards code clone detection in SourcererCC [40]. To our knowledge, SourcererCC is the best scaling code clone detection tool able to detect Type-3 clones. We demonstrate improvements in execution time relative to SourcererCC, while obtaining the same accuracy, for many common similarity thresholds, when evaluated on a large scale inter-project source code corpus [79].

We have presented a separate novel idea, the effective application of our technique to code clone search, without modification, in addition to code clone detection. Code clone search is a related problem to code clone detection where the user specifies a single code snippet (i.e. query block) to search for in a corpus of many code snippets. Once indexed, the corpus should be able to serve numerous such queries. Ours is among few techniques that can be applied to both of these problems. The contributions of our research work are the following:

- A novel code clone detection technique that can scale to very large scale source code repositories (or sets of repositories) with the ability to detect Type-1, Type-2, and Type-3 code clones, while maintaining high precision and recall.
- An extension of our proposed technique so that it can be effectively utilized for code clone search without modification.

3.2 Problem Formulation of Code Clone Search

Two syntactically or semantically equivalent code snippets are called code clone to each other. This pair of code snippets are called clone pairs and are denoted as $P(CB1, CB2, t)$ where $CB1$ and $CB2$ are similar code snippets and t is the type of clone [40].

Given a very large source code repository R, that contains collections of source code snippets $(CB1, CB2, CB3, CB4, ..., CBn)$ and a single query code snippet CQ, code clone search technique extracts each pair of code clones $P(CB, CQ)$. In other words, code clone search technique extracts the code snippets $(CB1, CB2, ..., CBk)$ where each of these code snippets CB is code clone of query code snippet CQ.

3.3 Code Clone Detection

Code clone detection is a well-known software engineering problem that aims to detect all the groups of code snippets that are functionally equivalent in a code base. It has numerous and wide ranging important uses in areas such as software metrics, plagiarism detection, aspect mining, copyright infringement investigation, code compaction, virus detection, and detecting bugs [63]. A scalable code clone detection technique, able to process large source code repositories, is crucial in the context of multi-project or Internet-scale code clone detection scenarios.

3.3.1 Problem Formulation of Code Clone detection

Given a pair of code snippets $CB1$ and $CB2$, and a similarity function f, a threshold value θ, a code clone detection technique detects whether this pair of code snippets, $P(CB1,CB2)$ is code clone or not.

In other words, given two large source code repositories, R_x and R_y, a similarity

function f and a threshold value θ, a code clone detection technique finds all the code snippet pairs or set of code snippet pairs $CB1$ and $CB2$ such that $f(CB1, CB2) \geq \lceil \theta \cdot max(|CB1|, |CB2|) \rceil$. For intra source code repository similarity R_x and R_y are same [40].

3.4 Adaptive Prefix Filtering Technique

Our proposed technique, centers on the combination of three filtering heuristics, two of which have been previously proposed for token-based code clone detection: *prefix filtering*, *token position filtering* and *adaptive prefix filtering*. Prefix filtering was proposed by researchers so that number of candidate pairs in token based code clone detection can be significantly reduced, improving performance and scalability. As a complement to prefix filtering, token position filtering utilizes the position of the tokens in the code snippet to further reduce the number of candidate pairs. The third heuristic, adaptive prefix filtering is an extended version of prefix filtering that looks for beneficial opportunities to filter candidate pairs even more aggressively, aimed at even further improving performance and scalability to large datasets. SourcererCC [40] has implemented the first two heuristics, while in our proposed code clone detection technique *we have applied all three heuristics*. In this section, we describe the context for token-based code clone detection, followed by an explanation of each of the three filtering heuristics. As a running example that helps clearly explain our approach we use 5 code snippets shown in Table 2.

For a token-based approach of code clone detection, at first, we must convert the source code into a set of tokens. To this end, we extract the set of string literals, keywords, and identifiers in each code snippet, removing all the special characters, operators and comments. Each of the extracted tokens can occur several times in a code snippet so we annotate each token with it's local occurrence frequency. The sum

```
//Code Snippet 1 (CB1)

public static int
 factorial(int result)
 {
  if(result <= 1) return 1;
  return result * factorial(result-1);
 }
```

```
//Code Snippet 2 (CB2)

public static int
 factorial(int n)
 {
  int result = 1;
  for(int i=1; i<=n; i++)
 {
     result = result * i;
  }
  return result;
  }
```

```
//Code Snippet 3 (CB3)

public static int
   factorial(int n)
{
   if(n >= 0)
   {
     result[0] = 1;
     for(int i=1; i<=n; i++)
    {
        result[i]=i * result[i-1];
     }
     return result[n];
    }
 }
```

```
//Code Snippet 4 (CB4)

public static void main
 (String[] args)
{
   int result = 5;
   int factorial = result;
   for(int i=result-1; i>1; i--)
   {
     factorial = factorial * i;
   }
}
```

```
//Code Snippet 5 (CB5)

public int factorial(int result)
{
   if(result == 0)
   {
     return 1;
   }
   else
 {
     return result * factorial(result-1);
   }
}
```

Table 2.: Code snippets for running example.

of the local occurrence frequencies for a specific token across all the code snippets in the corpus is it's global occurrence frequency. We sort all the tokens, in each code snippet, based their global frequency in ascending order. When there is a tie, the tokens are arranged in alphabetical order. For instance, in Table 3 we show the sorted token-based representation of the code snippets from Table 2.

3.4.1 Prefix Filtering

According to this heuristic, if the sorted tokens (as shown in Table 3) of a pair of code snippets *match at least one token in their prefix* then these snippets are a code

Code Snippet	Sorted Tokens
CB_1	$if^{\{9\}}_{\{1,3\}}$ $static^{\{10\}}_{\{1,4\}}$ $public^{\{11\}}_{\{1,5\}}$ $return^{\{13\}}_{\{2,6\}}$ $factorial^{\{14\}}_{\{2,9\}}$ $1^{\{15\}}_{\{3,12\}}$ $int^{\{17\}}_{\{2,14\}}$ $result^{\{18\}}_{\{4,19\}}$
CB_2	$for^{\{8\}}_{\{1,3\}}$ $static^{\{10\}}_{\{1,4\}}$ $public^{\{11\}}_{\{1,5\}}$ $n^{\{12\}}_{\{2,6\}}$ $return^{\{13\}}_{\{1,6\}}$ $factorial^{\{14\}}_{\{1,9\}}$ $1^{\{15\}}_{\{2,12\}}$ $i^{\{16\}}_{\{4,14\}}$ $int^{\{17\}}_{\{4,14\}}$ $result^{\{18\}}_{\{4,19\}}$
CB_3	$0^{\{7\}}_{\{2,3\}}$ $for^{\{8\}}_{\{1,3\}}$ $if^{\{9\}}_{\{1,3\}}$ $static^{\{10\}}_{\{1,4\}}$ $public^{\{11\}}_{\{1,5\}}$ $n^{\{12\}}_{\{4,6\}}$ $return^{\{13\}}_{\{1,6\}}$ $factorial^{\{14\}}_{\{1,9\}}$ $1^{\{15\}}_{\{3,12\}}$ $i^{\{16\}}_{\{6,14\}}$ $int^{\{17\}}_{\{3,14\}}$ $result^{\{18\}}_{\{4,19\}}$
CB_4	$5^{\{1\}}_{\{1,1\}}$ $String^{\{2\}}_{\{1,1\}}$ $args^{\{3\}}_{\{1,1\}}$ $main^{\{5\}}_{\{1,1\}}$ $void^{\{6\}}_{\{1,1\}}$ $for^{\{8\}}_{\{1,3\}}$ $static^{\{10\}}_{\{1,4\}}$ $public^{\{11\}}_{\{1,5\}}$ $factorial^{\{14\}}_{\{3,9\}}$ $1^{\{15\}}_{\{2,12\}}$ $i^{\{16\}}_{\{4,14\}}$ $int^{\{17\}}_{\{3,14\}}$ $result^{\{18\}}_{\{3,19\}}$
CB_5	$else^{\{4\}}_{\{1,1\}}$ $0^{\{7\}}_{\{1,3\}}$ $if^{\{9\}}_{\{1,3\}}$ $public^{\{11\}}_{\{1,5\}}$ $return^{\{13\}}_{\{2,6\}}$ $factorial^{\{14\}}_{\{2,9\}}$ $1^{\{15\}}_{\{2,12\}}$ $int^{\{17\}}_{\{2,14\}}$ $result^{\{18\}}_{\{4,19\}}$

Table 3.: Tokenized code snippets sorted based on global token ordering. Each token x is represented as $x^{\{m\}}_{\{l,g\}}$ where m is the global position (the position after sorting all the tokens of all the code snippets based on global frequency), l is the local frequency of the token in the code snippet, while g is the global frequency of token x in the corpus

clone candidate pair, which we verify whether are really code clone to each other in a subsequent verification step. On the other hand, if not a single token is matched, then we can discard them from further consideration.

Given two code snippets $CB1$ and $CB2$ and threshold value θ we calculate the the prefix of each code snippet is of size $|t| - \lceil \theta |t| \rceil + 1$ where $|t|$ is the total number of tokens in the corresponding code snippets. Now, we only test whether at least one token is matched in the two prefixes to determine if they need further scrutiny or if they can be filtered out.

3.4.1.1 Property

We generalize this to Property 1 as follows.

Property 1: *If two code snippets CB_1 and CB_2 consist of t terms each, which follow an order O, and if $|CB_1 \cap CB_2| \geq i$, then the sub-block CB_{sb1} consisting of the first $t - i + 1$ terms of CB_1 and the sub-block CB_{sb2} consisting of first $t - i + 1$ terms of CB_2 must match at least one token.*

3.4.1.2 Related Examples

In Table 4 we show the prefixes of two code snippets from our running example, assuming a similarity threshold value θ=0.8. For CB_1, whose size is $|t| = 16$, Table 4 shows that it's prefix is computed as $|t| - \lceil \theta |t| \rceil + 1 = 16 - 0.8 * 16 + 1 = 4$. In the fourth column of this table all the extracted tokens within the prefix of CB_1 are shown. It is worth mentioning that although the prefix length is four we have extracted five tokens because this metric ignores repeated tokens in the last position. Since *return* (last token with 1-prefix length) has repeated twice in CB_1 it is similarly included in the prefix.

Code Snippet	**Size of Code Snippet**	**1-Prefix Scheme**	**Tokens within Prefix**
CB_1	$\|t\| = 16$	$\|t\| - \lceil \theta \|t\| \rceil + 1 = 16 - 0.8 * 16 + 1 = 4$	{if static public return return}
CB_2	$\|t\| = 21$	$\|t\| - \lceil \theta \|t\| \rceil + 1 = 21 - 0.8 * 21 + 1 = 5$	{for static public n n}

Table 4.: Prefix filtering of CB_1 and CB_2

3.4.2 Token Position Based Filtering

For the second heuristic, token position filtering, we derive an upper bound by summing of the number of current matched tokens and minimum number of unseen tokens between two code snippets. If this upper bound is smaller than the needed threshold we can safely reject this code snippet pair.

3.4.2.1 Property

We derive property 2 for token position filtering, as follows.

Property 2: *Let snippets CB_1 and CB_2 be ordered and $\exists$ token t at index i in CB_1, such that CB_1 is divided in to two parts, where $CB_1(first) = CB_1[1...(i-1)]$ and $CB_1(second) = CB_1[i...(|CB_1|)]$. Now if $|CB_1| \cap |CB_2| \geq \theta * max(|CB_1|, |CB_2|)$,*

then $\forall\ t \in (CB_1 \cap CB_2)$, $|CB_1(first) \cap CB_2(first)| + min(|CB_1(second)|, |CB_2(second)|) \geq \theta * max(|CB_1|, |CB_2|)$.

3.4.2.2 Related Examples

For code snippets CB_1 and CB_2, if we examine the position of the first matching token *static*, we see that it is in position 2 for both code snippets. At position 2, the minimum number of unseen tokens is 14 for code snippet CB_1 and 19 for code snippet CB_2, since the total number of tokens is 16 and 21 for code snippets CB_1 and CB_2 respectively. We compute the upper bound between CB_1 and CB_2 as $\{1 + min(14, 19)\} = 15$, to communicate the number of matching tokens, in the best case scenario, given the position of the match in the *static* token. The upper bound in this case is lower than the required number of tokens (17) we needed to match, assuming the similarity threshold value of 0.8 (θ*$\{length(CB_1), length(CB_2)\}$=$0.8 * max(16, 21) = 17$). Therefore, we can reject this pair of code snippets without proceeding further.

3.4.3 Adaptive Prefix Filtering

Adaptive prefix filtering heuristic is the extended version of prefix filtering where rather than matching one token in the prefixes of code snippets, we match more, while deepening the size of the prefixes. For example, instead of matching only the token *static* in the prefix of CB_1 and CB_2 we can match multiple tokens. This variant defines an ℓ-prefix (instead of a 1-prefix scheme), where ℓ is the number of tokens we want to match. The size of the prefix (i.e. number of tokens within prefix) changes from $|t| - \lceil\theta|t|\rceil + 1$ to $|t| - \lceil\theta|t|\rceil + \ell$. Adaptive prefix filtering reduces the number of candidates more aggressively, at the cost of more comparisons in the filtering step. An advantageous value of ℓ can be selected based on the cost calculation framework

for each of the code snippet, discussed in Section 3.5.2.

3.4.3.1 Related Examples

In Table 5 we have shown how the 1-prefix scheme in Table 4 can be prolonged to 2-prefix and 3-prefix schemes. For instance, for CB_1 in Table 5 the length of 2-prefix scheme is $|t| - \lceil \theta|t| \rceil + 2 = 16 - 0.8 * 16 + 2 = 5$ resulting in the shown set of tokens.

Code Block	**2-Prefix Scheme**	**Tokens within 2-Prefix Scheme**	**3-Prefix Scheme**	**Tokens within 3-Prefix Scheme**
CB_1	$\|t\| - \lceil \theta\|t\| \rceil + 2 = 16 - 0.8 * 16 + 2 = 5$	{if static public return return factorial factorial}	$\|t\| - \lceil \theta\|t\| \rceil + 3 = 16 - 0.8 * 16 + 3 = 6$	{if static public return return factorial factorial 1 1 1}
CB_2	$\|t\| - \lceil \theta\|t\| \rceil + 2 = 21 - 0.8 * 21 + 2 = 6$	{for static public n n return}	$\|t\| - \lceil \theta\|t\| \rceil + 3 = 21 - 0.8 * 21 + 3 = 7$	{for static public n n return factorial}

Table 5.: Adaptive prefix filtering for CB_1 and CB_2

Here, *factorial* is the new token added in 2-prefix scheme which was not included in 1-prefix scheme. For the 2-prefix scheme of CB_1 and CB_2, the number of similar tokens between the prefixes is 3. These similar tokens are *static*, *public* and *return*. It is worth mentioning that although *return* has repeated two times in CB_1 it has occurred only once in CB_2 so we have to count it only once. According to adaptive prefix filtering, for CB_1 if we take 2-prefix scheme then we will keep CB_1 and CB_2 as candidate pairs because at least ℓ=2 tokens are matched in the prefixes of CB_1 and CB_2.

3.4.3.2 Property and Lemma

This is generalized in Property 3 for adaptive prefix filtering and Lemma 1, which assert that we can freely use an ℓ-prefix scheme in place of a 1-prefix scheme.

Property 3: *If snippets CB_1 and CB_2 consisting of t terms each, which follow an order O, and if $|CB_1 \cap CB_2| \geq i$, then the sub-block CB_{sb1} consisting of the first*

$t - i + \ell$ terms of CB_1 and the sub-block CB_{sb2} consisting of first $t - i + \ell$ terms of CB_2 will match at least ℓ tokens.

Lemma 1: *For any code snippet pair (CB_1, CB_2) if $P_\ell(CB_1) \cap P_\ell(CB_2) < \ell$ then $|CB_1 \cap CB_2| < \lceil \theta |t| \rceil$ where $t = max\{length(CB_1), length(CB_2)\}$ and θ is user defined threshold value [47].*

Here $P_\ell(CB_1)$ and $P_\ell(CB_2)$ denote the ℓ-prefix set of CB_1 and CB_2 which are the subsets of CB_1 and CB_2 respectively, and where each subset consists of the first $|t| - \lceil \theta |t| \rceil + \ell$ elements.

3.5 System Design for Adaptive Prefix Filtering

In this section, we describe how a system that uses adaptive prefix filtering can be implemented in practice, which is crucial to making this technique useful. A *delta inverted index* presents an efficient data structure for clone candidate filtering, while *prefix cost calculation* is important in determining the size of the prefix ℓ that optimizes the tradeoff between greater reduction in candidate pairs and the added cost of deeper filtering. Both the data structure and cost calculation are crucial steps in implementing the adaptive prefix filtering heuristic in practice.

3.5.1 Delta Inverted Index

An inverted index data structure is commonly used to retrieve matching documents (i.e. code snippets) using a particular token as a query (implemented in popular tools such as Apache Lucene [81]). For prefix filtering and token position filtering only a single inverted index data structure is required. Instead of creating index for each document, inverted index is created based on each token where it stores all the documents which contain that particular token. That's why it is named as inverted index. However, adaptive prefix filtering requires a separate inverted index

for each of the ℓ prefix schemes. A delta inverted index [47] overcomes the repetition that would occur if a simple set of inverted indices were used for adaptive prefix filtering. We describe this data structure.

The requirement for inverted list $I_\ell(e)$ is to store all the code snippets whose ℓ-prefix set contains the token e. Similarly, $I_{\ell+1}(e)$ stores all code snippets whose $\ell+1$ prefix set contains e, and $I_\ell(e) \subseteq I_{\ell+1}(e)$. A delta inverted index $\Delta I_{\ell+1}(e)$ data structure eliminates repetition by only storing the different code snippets between $I_\ell(e)$ and $I_{\ell+1}(e)$. At the outset, inverted index $\Delta I_1(e) = I_1(e)$, and as l increases we create delta inverted indexes $\Delta I_2(e)$, $\Delta I_3(e)$, ..., $\Delta I_t(e)$ for $I_1(e)$, $I_2(e)$, $I_3(e)$, ..., $I_t(e)$ where ($1 \leq \ell \leq$t-1).

The delta inverted index up to $\ell = 3$ is shown in Figure 3 for threshold value 0.8. Each of the three large boxes, ΔI_1, ΔI_2 and ΔI_3, represents a delta inverted index that can be queried separately, which contains a set of tokens mapped to their respective containing code snippets. ΔI_2 contains code snippets that are not present in I_1 but present in I_2. Index ΔI_3 contains code snippets that are not in I_1 and I_2 but are present in I_3. As an example, for token *return* inverted index $I_1(return)$ contains CB_1 and inverted index $I_2(return)$ contains CB_1, CB_2 and CB_5, while delta inverted index $\Delta I_2(return)$ contains CB_2 and CB_5.

3.5.2 Cost Calculation

In order to select the appropriate prefix length for each of the code snippets, we need to optimize the trade-off between filtering cost (i.e. the cost of looking up tokens deeper in the delta inverted index) and verification cost (i.e. the cost of determining if a pair of code snippets are actual clones by comparing all of the necessary tokens). The adaptive prefix filtering technique iteratively estimates the cost for ℓ-prefix scheme and $\ell+1$-prefix scheme, and if $\ell+1$-prefix scheme's cost is

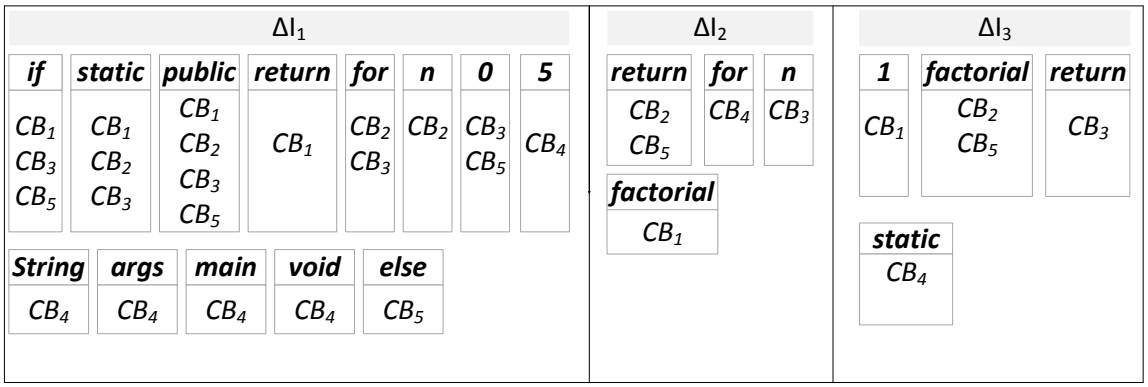

Fig. 3.: *Delta inverted index data structure for CB_1 to CB_5*

greater than ℓ-prefix scheme cost then we select ℓ-prefix scheme for that particular code snippet. Otherwise we continue to compute the next prefix scheme's cost. Prior results include the fact that this technique selects a global minimum for the optimal prefix scheme [47]. Algorithm 1 lists steps of cost calculation which is integral part of adaptive prefix filtering technique. Without cost calculation this technique is not adaptive at all.

Suppose that for each code snippet $CB \in R$, where R is the repository of all code snippets, $F_\ell(CB)$ is the filtering cost and $V_\ell(CB)$ is verification cost. Therefore, for code snippet CB, we can derive following general equation for the cost:

$$Total\,Cost_\ell = F_\ell(CB) + V_\ell(CB) \tag{3.1}$$

For each code snippet $CB \in R$, it is necessary to query the inverted list of each token $e \in P_\ell(CB)$, where $P_\ell(CB)$ denotes the prefix set for the ℓ prefix scheme for code snippet CB. We introduce a new notation $\Phi_\ell(CB)$ to denote the set of all delta inverted indices for ℓ-prefix scheme used in the filtering step of code snippet CB such that $\Phi_\ell(CB) = \{\Delta I_i(e) | e \in P_\ell(CB), 1 \leq i \leq \ell\}$. Similarly, let $\Delta\Phi_\ell(CB)$ denote the set of additional delta inverted lists to be processed in ℓ prefix scheme comparing

to $\ell - 1$ prefix scheme. Assume $C_\ell(CB)$ is the candidate set of code snippet CB, containing the code snippets that appear at least ℓ number of inverted lists of the elements in $P_\ell(CB)$.

Also, let $cost_v(CB)$ be the average cost of verifying the candidate CB. Using these abstractions, we can derive filter cost and verification cost using the following equations:

$F_\ell = \sum_{e \in P_\ell(CB)} |I_\ell(e)|$

$V_\ell = cost_v(CB) \cdot |C_\ell(CB)|$

As delta inverted index contains only the different code snippets in between ℓ-prefix scheme and $\ell + 1$-prefix scheme, we do not need to calculate the filtering cost from scratch every time, converting the previous filter cost equation to the following variant.

$F_\ell(CB) = F_{\ell-1}(CB) + \sum_{\Delta I(e) \in \Delta\Phi_\ell(CB)} |\Delta I(e)|$

Prefix Scheme	Filter Cost
1-Prefix	$\|I_1(if)\|$+ $\|I_1(static)\|$+ $\|I_1(public)\|$+ $\|I_1(return)\|$ =3+3+4+1=11
2-Prefix	$\|I_1(if)\|$+ $\|I_1(static)\|$+ $\|I_1(public)\|$+ $\|I_1(return)\|$+ $\|I_1(factorial)\|$+ $\|\Delta I_2(if)\|$+ $\|\Delta I_2(static)\|$+ $\|\Delta I_2(public)\|$ + $\|\Delta I_2(return)\|$+ $\|\Delta I_2(factorial)\|$ =11+2+1=14
3-Prefix	$\|I_1(if)\|$+ $\|I_1(static)\|$+ $\|I_1(public)\|$+ $\|I_1(return)\|$+ $\|I_1(factorial)\|$+$\|I_1(1)\|$ + $\|\Delta I_2(if)\|$+ $\|\Delta I_2(static)\|$+ $\|\Delta I_2(public)\|$ + $\|\Delta I_2(return)\|$+ $\|\Delta I_2(factorial)\|$+ $\|\Delta I_2(1)\|$+$\|\Delta I_3(if)\|$ + $\|\Delta I_3(static)\|$+ $\|\Delta I_3(public)\|$+ $\|\Delta I_3(return)\|$ + $\|\Delta I_3(factorial)\|$+ $\|\Delta I_3(1)\|$ =14+1+1+2+1=19

Table 6.: Calculation of filter cost for CB_1

In Table 6 we show the calculation of filter cost for different prefix schemes for our running example, CB_1. For instance, to calculate the filter cost for prefix scheme $\ell = 2$ we can use filter cost from prefix scheme $\ell = 1$. So the filter cost for prefix scheme $\ell = 2$ (14) is the summation of filter cost of prefix scheme $\ell = 1$ (i.e. 11) and the size of the delta inverted index of the tokens: {*if*, *static*, *public*, *return*, *factorial*}.

Algorithm 1 Pseudo-code of the cost calculation for each code snippet.

Input: CB=code snippet represented as a bag-of-tokens with tokens sorted according to their global frequency, $\Delta\Phi_\ell$(CB)=set of additional delta inverted indices for ℓ-prefix scheme comparing to (ℓ-1)-prefix scheme of code snippet CB , $\Phi_\ell(CB)$=$\{\Delta I_i(e)|e \in P_\ell(CB), 1 \leq i \leq \ell\}$ where $\Delta I_i(e)$=delta inverted index of i-prefix scheme, n=user-defined maximum threshold scheme

Output: Total Cost of code snippet CB, prefix scheme ℓ with lowest cost for code snippet CB

Variables: T_ℓ=Total Cost of ℓ-prefix scheme; F_ℓ=Filter Cost of ℓ-prefix scheme; V_ℓ=Verification Cost of ℓ-prefix scheme; $C_\ell(CB)$=candidate set of ℓ-prefix scheme; $H[CB]$= Hashmap storing the number of processed lists that contain code snippet CB; S=union set of multiple $\Delta I_\ell(e)$; $cost_v(CB)$=average cost of verifying the candidate CB; $C^{=}_{\ell-1}(CB)$=set of code snippets that occur at least $(\ell-1)$ lists in $\Phi_{\ell-1}(CB)$; $C^{>}_{\ell-1}(CB)$=set of code snippets that appear in more than $(\ell-1)$ lists in $\Phi_{\ell-1}(CB)$;

1: **function** COST_CALCULATION(CB,$\Delta\Phi_\ell$(CB))
2: $H[CB] = 0$
3: ℓ=1
4: S=$\emptyset$
5: **while** (ℓ <= n) **do**
6: $C^{=}_{\ell-1}(CB)$=$\emptyset$
7: $C^{>}_{\ell-1}(CB)$=$\emptyset$
8: $tokensToBeIndexed = |CB| - \lceil\theta|CB|\rceil + \ell$
9: **for** each token e∈CB[1:$tokensToBeIndexed$] **do**
10: F_ℓ=$F_\ell(CB) = F_{\ell-1}(CB) + \sum_{\Delta I(e)\in\Delta\Phi_\ell(CB)} |\Delta I(e)|$ ▷ calculation of filter cost
11: **if** (ℓ == 1) **then**
12: $C_\ell(CB) = C_\ell(CB) \cup \Delta I_\ell(e)$
13: **for** each code snippet $CB \in \Delta I_\ell(e)$ **do**
14: $H[CB] = H[CB] + 1$
15: **end for**
16: **else**
17: $S = S \cup \Delta I_\ell(e) \in \Delta\Phi_\ell(CB)$
18: **for** each code snippet $CB \in S$ **do**
19: **if** ($H[CB] > (\ell-1)$) **then**
20: $C^{>}_{\ell-1}(CB)$= $C^{>}_{\ell-1}(CB) \cup CB$
21: **end if**
22: **if** ($H[CB] == (\ell-1)$) **then**
23: $C^{=}_{\ell-1}(CB)$= $C^{=}_{\ell-1}(CB) \cup CB$
24: **end if**
25: **end for**
26: **for** each $\Delta I_\ell(e) \in \Delta\Phi_\ell(CB)$ **do**
27: **for** each code snippet $CB \in \Delta I_\ell(e)$ **do**
28: $H[CB] = H[CB] + 1$
29: **end for**
30: **end for**
31: $|C_{(\ell)}(CB)| = |C^{>}_{(\ell-1)}(CB)| + |C^{=}_{(\ell-1)}(CB) \cap \bigcup_{\Delta I(e)\in\Delta\Phi_\ell(CB)} \Delta I(e)|$
32: **end if**
33: **end for**
34: V_ℓ=$cost_v(CB) \cdot |C_\ell(CB)|$ ▷ calculation of verification cost
35: T_ℓ= F_ℓ + V_ℓ ▷ calculation of total cost
36: **if** ($T_\ell > T_{\ell-1}$) **then return** $T_{\ell-1}$, ($\ell-1$)
37: **end if**
38: ℓ++
39: **end while**
40: **end function**

The average cost of verifying a code snippet CB is $cost_v(CB)$. We compute this cost using s_u and s_l, the upper bound and lower bound of the sizes of all the code snippets within code repository R. We express this via following equation:

$$cost_v(CB) = |CB| + \frac{s_{|u|} + s_{|l|}}{2}$$

The average cost of verifying CB_1, $cost_v(CB_1)$=16+$\frac{16+28}{2}$=38. Here, 16 and 28 are the lower and upper bound of the size among the five code snippets in our running example.

To estimate the candidate set size of 1-prefix scheme, $C_1(CB)$, we simply calculate the number of code snippets which appear in at least one inverted list using the tokens within 1-prefix scheme. In Table 7, the first row shows that CB_1,CB_2,CB_3,CB_5 have appeared in at least one inverted index of tokens within 1-prefix scheme of CB_1, and, therefore, the candidate set size is 4. For computing the candidate set size $|C_{\ell+1}(CB)|$ of $(\ell+1)$-prefix scheme we can utilize the candidate set size of ℓ-prefix scheme $|C_\ell(CB)|$.

Those code snippets that appear in more than ℓ number of inverted lists of tokens within ℓ-prefix scheme, also appear in the candidate set of $(\ell+1)$-prefix scheme. All other code snippets which appear in at least ℓ number of inverted lists of tokens within ℓ-prefix scheme can also appear in the candidate set of $(\ell+1)$-prefix scheme if and only if these code snippets appear in the additional delta inverted lists of the $(\ell+1)$-prefix scheme. We take the summation of the size of these two sets where one set contains the intersection of the code snippets appearing both in the ℓ-number of inverted lists in ℓ-prefix scheme and the additional delta inverted lists of $(\ell+1)$ prefix scheme and the other set contains the code snippets appearing in more than ℓ number of inverted lists in ℓ-prefix scheme.

Let $C_\ell^=(CB)$ represents the set of code snippets that occur at least ℓ list in $\Phi_\ell(CB)$ and let $C_\ell^>(CB)$ represents the set of code snippets that appear in more than ℓ lists in $\Phi_\ell(CB)$. Using these, we can define the following candidate set equation.

$$|C_{(\ell+1)}(CB)| = |C_\ell^>(CB)| + |C_\ell^=(CB) \cap \bigcup_{\Delta I(e) \in \Delta\Phi_{\ell+1}(CB)} \Delta I(e)|$$

In Table 7 we show how we utilize the candidate set size $|C_1(CB_1)|$ of 1-prefix scheme to derive the candidate set size $|C_2(CB_1)|$ of 2-prefix scheme for code snippet CB_1. For estimating candidate set size $|C_2(CB_1)|$ first we have to calculate the number of code snippets which appear in more than one inverted lists of tokens within the 1-prefix scheme. In this case, we only need to consider those code snippets which appear in ΔI_2 which is the delta inverted list for 2-prefix scheme of CB_1. If we check all the tokens {*if, static, public, return, factorial*} within 2-prefix scheme of CB_1, we see only tokens {*return ,factorial*} are in CB_1,CB_2,CB_5 in the delta inverted list ΔI_2. Since CB_1,CB_2,CB_5 have appeared in at least 2 inverted lists of the tokens within 1-prefix scheme of CB_1, these code snippets are the elements of set $C_1^>(CB_1)$. Therefore code snippets CB_1,CB_2,CB_5 should be included in the candidate set $C_2(CB_1)$ of 2-prefix scheme of CB_1.

Those code snippets which appear only in one inverted list of the tokens of 1-prefix scheme, are the elements of set $C_1^=(CB_1)$. If those code snippets of $C_1^=(CB_1)$, also appear in the additional delta inverted list $\Delta\Phi_2(CB_1)$ of 2-prefix scheme then these code snippets are the elements of the candidate set $C_2(CB_1)$ of 2-prefix scheme. Therefore we have to take the intersection between two sets where one set, $C_1^=(CB_1)$, contains the code snippets appearing in one inverted list and the other set contains the code snippets appearing in the additional delta inverted list $\Delta\Phi_2(CB_1)$. For CB_1

there are no code snippets that appear only in one inverted list, and the intersection of these two sets is $\emptyset$.

Prefix Scheme	Candidate Set Size
1-Prefix	$\{\|\Delta I_1(if)\|, \|\Delta I_1(static)\|, \|\Delta I_1(public)\|, \|\Delta I_1(return)\|\}=$ $\{\|(CB_1, CB_3, CB_5)\|,\|(CB_1, CB_2, CB_3)\|$ $\|(CB_1, CB_2, CB_3, CB_5)\|,\|(CB_1)\|\}=$ $\{\|(CB_1, CB_2, CB_3, CB_5)\|\ \}=4$
2-Prefix	$\|C_2(CB_1)\| = \|C_1^{>}(CB_1)\| + \|C_1^{=}(CB_1) \cap \bigcup_{\Delta I(e)\in\Delta\Phi_2(CB_1)} \Delta I(e)\|$ $= \|C_1^{>}(CB_1)\| + \|C_1^{=}(CB_1) \cap \{\Delta I_2(if) \cup \Delta I_2(static) \cup \Delta I_2(public) \cup$ $\Delta I_2(return) \cup \Delta I_2(factorial) \cup \Delta I_1(factorial)\}\|$ $=\|C_1^{>}(CB_1)\| + \|C_1^{=}(CB_1) \cap \{CB_2 \cup CB_5 \cup CB_1\}\|$ $=\|C_1^{>}(CB_1)\| + \|\emptyset \cap \{CB_2 \cup CB_5 \cup CB_1\}\|$ $=\|\{CB_1, CB_2, CB_5\}\| + \|\emptyset \cap \{CB_2 \cup CB_5 \cup CB_1\}\|$ $=3 + 0 = 3$
3-Prefix	$\|C_3(CB_1)\| = \|C_2^{>}(CB_1)\| + \|C_2^{=}(CB_1) \cap \bigcup_{\Delta I(e)\in\Delta\Phi_3(CB_1)} \Delta I(e)\|$ $= \|C_2^{>}(CB_1)\| + \|C_2^{=}(CB_1) \cap \{\Delta I_3(if) \cup \Delta I_3(static) \cup \Delta I_3(public) \cup$ $\Delta I_3(return) \cup \Delta I_3(factorial) \cup \Delta I_3(1) \cup \Delta I_2(1) \cup \Delta I_1(1)\}\|$ $=\|C_2^{>}(CB_1)\| + \|C_2^{=}(CB_1) \cap \{CB_4, CB_3, CB_2, CB_5, CB_1\}\|$ $=\|C_2^{>}(CB_1)\| + \|\emptyset \cap \{CB_4, CB_3, CB_2, CB_5, CB_1\}\|$ $=\|\{CB_1, CB_2, CB_3, CB_5\}\| + \|\emptyset \cap \{CB_4, CB_3, CB_2, CB_5, CB_1\}\|$ $=4 + 0 = 4$

Table 7.: Calculation of candidate set size for CB_1

Prefix Scheme	Filter Cost	Verification Cost	Total Cost
1-Prefix	11	$cost_v(CB) \cdot \|C_1(CB)\|$ =38*4.0=152.0	163.0
2-Prefix	14	$cost_v(CB) \cdot \|C_2(CB)\|$ =38*3.0=114.0	128.0
3-Prefix	19	$cost_v(CB) \cdot \|C_3(CB)\|$ =38*4.0=152.0	171.0

Table 8.: Calculation of total cost for CB_1.

Finally we take the summation of the size of those two sets. In Table 8, we show the calculation of verification cost and total cost for our running example. To calculate the verification cost of CB_1 we have multiplied the average verification cost by the candidate set size. As the 2-prefix scheme cost is smaller than the 3-prefix scheme cost, we stop here and take $\ell = 2$ as the preferred prefix scheme.

3.5.3 Code Clone Search

The adaptive prefix filtering technique can also be utilized for code clone search, where a user-specified code snippet (i.e. the query) is matched in a corpus consisting of numerous code snippets. In code clone search, different from code clone detection,

we do not pre-specify the similarity threshold value before the index is built from the corpus, as we would like for the same index to be able to serve different queries with different threshold values. Therefore, the index structure should be able to deal with any threshold value, $1 <= |s| <= 10$, where $|s|$ is the maximum threshold value the index could serve. As a naive approach, we can build an index for all possible threshold value from 1 to $|s|$ for each code snippet. However, this would take up huge space and be very time consuming.

Instead of building delta inverted indices for each threshold value from 1 to l, we can build delta inverted indices for the maximum threshold value $|s|$, i.e. for 100% similarity (threshold value 10). With this maximum threshold value we build delta inverted indices, ΔI_1, ΔI_2, ΔI_3 and so on, until we reach the maximum prefix scheme. For example for code block $CB1$ from Table 3, ΔI_1 contains the token *if*, ΔI_2 contains the token *static*, ΔI_3 contains the token *public* and so on. We continue to populate the delta inverted indices until ΔI_8, which contains the final token for this specific code snippet, *result*. We continue this process for all of the code snippets in the corpus. At retrieval time, we use this data structure as the means to answer code clone search queries with retrieval-time similarity thresholds. Apart from this modification to the data structure, the algorithm follows the same logic as in code clone detection.

3.6 Experimental Results

Our goal is to implement a code clone detection and search tool that can scale to massive inter or intra project source code repositories and overcome limitations in many existing tools, such as, unsustainable execution time, inadequate system memory, restrictions in inner data structures, and exhibiting errors due to their design not expecting a large input [40, 43, 82]. In evaluating our tool, we focused on answering

the following set of research questions.

- **RQ 1:** *Does adaptive prefix filtering achieve better performance at scale than the best state of the art tool SourcererCC?*

  SourcererCC is a recent code clone detection tool aimed at scalability on a single machine [40]. The adaptive prefix filtering heuristic presented in our research work, extends the filtering heuristics used by SourcererCC, so a comparison between the two is both natural and necessary.

  Several code clone detection tools have been benchmarked in recent papers [83, 84]. Among all of the measured tools, four publicly available tools achieve exceptional scalability and accuracy in code clone detection: CCFinderX [36], Deckard [38], iClones [41] and NiCad [85]. In turn, SourcererCC has been measured to outperform these four publicly available and popular tools [40].

  Recently, several popular code clone detection tools were compared [46], including VUDDY [46], SourcererCC [40], CCFinderX [36] and Deckard [38]. SourcererCC outperformed CCFinderX [36] and Deckard [38]. VUDDY outperformed SourcererCC, however, it only detects Type-1 and Type-2 clones.

- **RQ 2:** *Does adaptive prefix filtering achieve reasonable accuracy in clone detection?*

  This research question aims to determine whether the adaptive prefix filtering heuristic can achieve reasonable precision and recall, and whether it can serve as a replacement to SorcererCC.

- **RQ 3:** *Does the application of adaptive prefix filtering to code clone search achieve acceptable query response time?*

Searching for similar code snippets in very large scale source code repository within reasonable amount of time is a challenging problem. In using our technique for code clone search, we want to determine whether this application produces reasonable response time, which is key for it to be useful in practice.

3.6.1 Performance of Adaptive Prefix Filtering (RQ1)

In this section we discribe a comparison of the scalability of adaptive prefix filtering technique relative to SourcererCC [40]. To answer this research question we rely on publicly available large-scale evaluation datasets, which have recently become available. The IJaDataset 2.0 [79] is a large inter-project Java repository containing 25,000 open source projects with 3 million source files and 250MLOC. It is mined from SourceForge and Google Code [40]. To compare SourcererCC with adaptive prefix filtering, both tools were benchmarked on a standard workstation with 3.50GHz quad-core i5 CPU, 32.0 GB of RAM memory, and 64-bit windows operating system. The execution of both tools was scripted to measure the run-time of both tools 5 times for each input and report the average. For adaptive prefix filtering, we used the 2-prefix scheme in constructing the delta inverted index. In order to measure the performance at different input sizes, we blindly selected a subset of the Java source files in IJADataset [79] ranging from 10,000 to 160,000 files, approximately from 1MLOC to 17MLOC, such that the selected files at smaller input sizes were also contained in the larger inputs. This ensures a stable measurement since code clone technique execution time may be reliant on clone density [40]. Both techniques consumed extreme amounts of time (>10 hours) at 160,000 files(approximately 17MLOC), and therefore we did not attempt using larger input sizes. And for same reason we limited similarity degree from 70% to 90% in comparison. But for smaller dataset(1MLOC) we have experimented on all similarity degrees(from 10% to 100%).

Figure 4 shows a comparison in execution time for adaptive prefix filtering technique and SourcererCC for threshold values 7, 8, and 9 (i.e. 70%-90% similarity). For threshold values 7 and 8 adaptive prefix filtering is showing an improvement relative to SourcererCC, but not at threshold value 9. As the the input size increases toward 160,000 files, the difference in execution time between SourcererCC and adaptive prefix filtering becomes wider for threshold values 7 and 8.

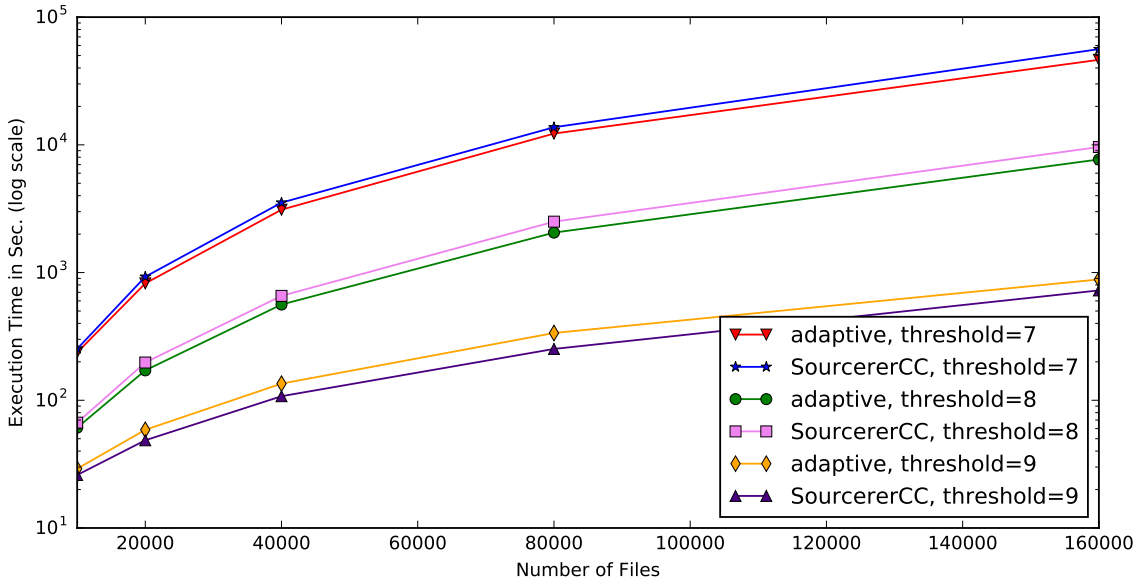

Fig. 4.: *Comparison of execution time for different input sizes between adaptive prefix filtering and SourcererCC.*

In Table 9 we show the execution times for SourcererCC and adaptive prefix filtering technique for a fixed input of 10,000 files(approximately 1MLOC), but at varying threshold values. We show both raw execution times as well as the percentage improvement of adaptive prefix filtering technique in the rightmost column. Adaptive prefix filtering performs better than SourcererCC for threshold values ranging from 4 to 8. For threshold value 1, 2, 3, 9 and 10 the execution time of adaptive prefix filtering technique is larger. We argue that the similarity degrees from 40% to 80%

Threshold Value	Adaptive Prefix Filtering	SourcererCC	% Improvement
1	4982.23	4452.31	-10.64%
2	5357.54	5339.58	-0.34%
3	4499.14	4478.93	-0.45%
4	3001.19	3015.76	0.49%
5	1627.95	1652.18	1.49%
6	712.45	740.16	3.89%
7	222.68	249.27	11.94%
8	58.84	64.25	9.19%
9	27.25	23.99	-11.96%
10	23.25	21.89	-5.85%

Table 9.: Comparison of execution time (in seconds) between adaptive prefix filtering and SourcererCC (10,000 files).

are more commonly used. Exact or near-exact similarity (i.e. 90% and 10%) do not make sense for detecting Type-2 and Type-3 code clones, while code clones detected with 10% to 30% similarity are likely to contain a very large number of false positives.

To better understand the effect of the more aggressive filtering performed by adaptive prefix filtering, and understand the rationale behind the performance numbers in Table 9, we compared the number of candidate pairs (in log scale) of SourcererCC and adaptive prefix filtering technique in Figure 5 for 10000 files. The candidate pairs are the number of remaining code clone candidates after the filtering that both techniques perform. A reduction in the number of candidates translates to improvement in execution time, after the penalty for the more sophisticated indexing structure and cost calculation of adaptive prefix filtering is factored in. We observe the strongest reduction in the percentage of candidate pairs in the similar thresholds of 4 to 9 where we observed improvements in execution time. Threshold value 9 has a large percentage reduction in candidate numbers, but the actual reduction numbers are very low[1] and insufficient for offsetting the penalty of more aggressive filtering. For threshold value 10 (100% similarity) the percentage decrease is 0. This is logical

[1]Note that it is a log scale graph.

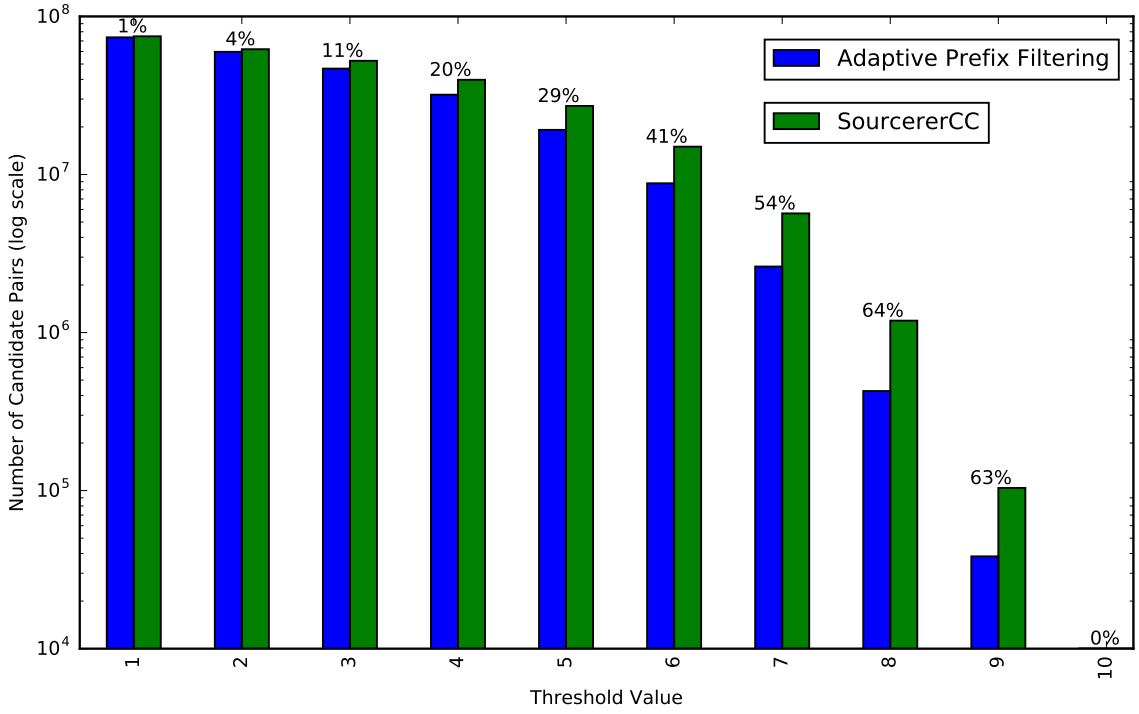

Fig. 5.: *Comparison of number of candidate pairs between adaptive prefix filtering and SourcererCC (10,000 files).*

because with complete similarity adaptive prefix filtering reduces to regular prefix filtering as there is no room to extract a deeper prefix.

Memory Usage. Fitting with the limited memory budget available on commodity machines is an important factor in scalable code clone detection. For our adaptive prefix filtering technique we use a deeper prefix scheme, resulting in a larger index that will clearly consume more memory than SourcererCC. We measured the memory requirement for both SourcererCC and adaptive prefix filtering for 1 MLOC in the indexing and clone detection phases. While varying the threshold value, we measured how much memory is consumed in each iteration of the indexing process and recorded the maximal amount of memory consumed. The resulting memory requirement was 103 MB for threshold value 1 and 60 MB for threshold value 10, which

is 60% and 20% higher relative to SourcererCC's maximal memory footprint in indexing. The memory usage is smaller for the larger threshold value because as the threshold value is increasing the prefix length is decreasing, so fewer tokens need to be stored in the index structure. In the subsequent clone detection phase we measured the memory requirement for each query block, again, selecting the maximum amount of memory required. We obtained 316 MB for adaptive prefix filtering and 204 MB for SourcererCC. In conclusion, although adaptive prefix filtering technique requires higher amount of memory compared to SourcererCC, the memory requirement of both techniques is relatively small and would not impact most deployments. Part of the reason for the small footprint is the efficient allocation of memory by Apache Lucene, which was used as to store the inverted index by both SourcererCC and our implementation.

3.6.2 Accuracy of Adaptive Prefix Filtering (RQ2)

We use precision and recall for measuring the accuracy of adaptive prefix filtering technique as these are the two most commonly used metrics used to determine the quality of a code clone detection technique [80]. Measurement of clone recall and precision has been greatly aided by recent datasets and frameworks like BigCloneEval [86]. This framework can be used for the evaluation of code clone detection tools based on the BigCloneBench clone detection benchmark [87]. BigCloneBench contains a large set of known clones from the inter-project software repository IJaDataset 2.0 [78, 40], which we used in RQ1. Note that SourcererCC and adaptive prefix filtering produce the same clones as output due to the inherent similarity in the techniques.

Recall. For measuring recall of adaptive prefix filtering we use file level granularity, which means the clone pairs are actually pair of two Java source files that are detected as code clones to each other. The evaluation of SourcererCC was at the

method level [40]. In measuring recall using BigCloneBench, Type-3 and Type-4 code clones are separated in four categories because it is difficult to separate Type-3 and Type-4 since there is no consent on the smallest similarity of Type-3 [78]. These four categories are: Very Strongly Type-3 (VST3) clones that has range of syntactical similarity from 90% to 100%, Strongly Type-3 (ST3) that has range of syntactical similarity from 70% to 90%, Moderately Type-3 (MT3) that has range of syntactical similarity from 50% to 70% and Weakly Type-3 (WT3/T4) that has range of syntactical similarity from 0% to 50%, which are often Type-4 code clones [40].

For this evaluation we used a similarity threshold of 70% in executing adaptive prefix filtering, which is the default setting for BigCloneEval. Our technique produced very high recall for Type-1 code clones (97%), and detected Type-2 code clones with a reasonable recall of 77%. For Type-3 clones, the recall decreases significantly, from 60% for the VST3 category, 26% for ST3, 16% for MT3 and less than 1% for WT3/T4. The Weakly Type-3/Type-4 clones have low syntactic similarity which makes it very hard for our technique to detect at the 70% threshold, so this result is not unexpected.

Precision. The adaptive prefix filtering technique detects the same code clones as SourcererCC, and therefore it's precision (and recall) will be same as SourcererCC. SourcererCC's precision was previously evaluated via a set of 390 clone pairs, which were manually identified by several researchers with high mutual inter-agreement. Out of these 390 clone pairs, 355 were true positives while 35 were found false positives, resulting in a precision of 91% computed at method level granularity [40].

The value of the precision metric, unlike recall, is influenced by the number of false positives. For token based code clone detection techniques, a common source of false positive clones stems from the fact that these techniques commonly treat the input as a bag of words, ignoring the ordering of tokens in the input. To illustrate this point we show a false positive clone pair identified by our technique (and likely most

other token based techniques), snippets of which are shown in Table 10. The pair of code snippets constituting the false positive example are: 1) a code snippet that implements password validation and 2) a code snippet that implements password encryption. Although these two code snippets have 70% similar tokens they are functionally dissimilar. In fact, the pair of code snippets have significant differences at the line level and their different purpose would be easily observed by a human. However, both of the code snippets are dealing with passwords there are significant number of similar tokens. Some of these are: *password*, *passwordInDb*, *MessageDigest*, *update*.

```
//Password Validation

byte[] digestInDb =
  new byte[pwdInDb.length - SALT_LENGTH];
System.arraycopy(pwdInDb, SALT_LENGTH, digestInDb, 0, digestInDb.length);
if (Arrays.equals(digest, digestInDb)) {
  return true;
} else {
  return false;
}
```

```
//Password Encryption

pwd = new byte[digest.length + SALT_LENGTH];
System.arraycopy(salt, 0, pwd, 0, SALT_LENGTH);
System.arraycopy(digest, 0, pwd, SALT_LENGTH, digest.length);
return byteToHexString(pwd);
```

Table 10.: Snippets of false positive clone pair identified by our technique.

3.6.3 Applicability Towards Code Clone Search (RQ3)

We showed that the adaptive prefix filtering technique can easily be extended so that it can support similarity search. In this section, we examine the practicality of code clone search based on this technique by measuring the performance of this variant. Our goal here is to show that the use of our technique for code clone search is practical, but not to show that our technique outperforms those that specialize solely on the code clone search problem. Table 11 shows the time it takes to construct the

index, and, more importantly, the average query time for 1000 files randomly selected from IJADataset with similarity degree 80% (i.e. threshold value 8). By index time, we refer to the time required to build the special indexing structure given a maximum similarity threshold of 100%. After creating this index, we use a random selection of 1000 Java source files, from the files used to construct the index, performing a code clone search for each file. We report the average search time for different input sizes.

In examining the performance of our technique in Table 11, we observe sub-second response times for each queries, even at the larger corpus sizes. This is likely to be reasonable performance for many applications.

Number of Files	Similarity	Index Time (in sec.)	Query Time (in sec.)
10000	80%	70.52	0.046
20000	80%	160.12	0.095
40000	80%	351.29	0.177
80000	80%	736.93	0.368

Table 11.: Performance of code clone search using adaptive prefix filtering.

3.7 Characterizing Duplicate Code Snippets between Stack Overflow and Tutorials

Nowadays, developers frequently consult online resources to learn new skills, expand or refresh their knowledge, or avoid repetitive tasks [88]. Code snippets (or code blocks) available on many sources of software development related information on the Web are easy to reuse and incorporate into existing projects. While reusing online code snippets improves the speed of development, it has possible negative side-effects on code quality and maintainability [89, 90]. For instance, reusing online code snippets is susceptible to introducing bugs and software vulnerabilities [91, 92] and exposing security risks as a result of outdated or poorly written code [12, 93]. Additionally, by copying and pasting code snippets with unknown origin, unaware

developers can cause license violations [94, 13].

One of the largest and most visited sources of reusable code snippets is Stack Overflow, a Q&A website with a large and active community of 9.9 million users, and a corpus of 17 million questions and 26 million answers[2]. Each Stack Overflow question pertains to a specific technical problem, and contains one or more answers that often include code blocks implementing a solution to the problem. While Stack Overflow provides answers to a large set of development problems and has a permissive license permitting reuse of posted code snippets by developers in their projects, studies show that a significant amount of posted code does not originate on this platform, but is reproduced from elsewhere [13, 10, 27].

Different from Stack Overflow, online tutorials provide step-by-step instructions on a specific development topic often introducing a practical application as a running example and including numerous code snippets accompanied by a rich and detailed description [95, 96]. Code snippets on tutorials are usually longer, and often several code snippets form a logical sequence interspersed with natural language explanations [20]. No tutorial source with the scale of Stack Overflow exists, and each small scale source uses its own licensing scheme.

In this study, we analyze the duplication of code snippets between tutorials and Stack Overflow, with the goal of 1) characterizing developer rationale behind reproducing snippets from source to source; and 2) understanding the scale and properties of duplicate snippets, including their evolution over time. Our findings lead towards better understanding of this phenomenon and how the two sources can be best engineered to clearly display the origin of snippets available for reuse by developers, and to improve the design of tools that mine code snippets.

[2]Data as of 21 January, 2019.

3.7.1 Research Methodology & Experimental Setup

In order to obtain easily discernible code snippets and their edit histories, we leverage the *SOTorrent* dataset, which provides the version histories for over 40 million posts, based on the official Stack Overflow data dumps, including 122 million text block versions and 77 million code snippet versions [97]. Stack Overflow data is distributed with the *Creative Commons Attribute - ShareAlike 3.0 Unported (CCBY-SA 3.0)* license. This indicates that developers must reference the original post when reusing code and must use a similar license for any derivative work.

Separately, we curated a list of 599 Android tutorials available on 5 popular software development related websites[1]. Some of the tutorial sites provided explicit licenses, while others used non-standard language or provided no specific license, as shown in Table 12. We extracted the code snippets from the Android tutorials based on a set of HTML tags, obtained by manually examining the patterns used to display snippets in each individual site. The HTML tags specifically focused on extracting Java code, ignoring other code blocks commonly present in Android tutorials, e.g., written in XML. As this approach failed to filter all non-Java code snippets, we used regular expressions and manual analysis to ensure that only Java code snippets remained. Following this filtering step, the tutorial corpus consisted of a total of 2,504 code snippets, ranging from 1 to 626 lines of code, and a median of 19 lines of code. Unlike the SOTorrent dataset, the edit histories of tutorials were not possible to reconstruct and, while some tutorials displayed dates of last modification, we found several examples where the dates were not updated and therefore unreliable.

In order to extract code snippets from Stack Overflow, we executed SQL queries on the BigQuery [98] interface to the SOTorrent dataset (2018-12-09 version) [97]. We filtered posts based on the android Stack Overflow tag, obtaining 3,114,844 code

Table 12.: Curated set of Android tutorials (as of 21 January, 2019).

Tutorial Source	**Number of Tutorials**	**Number of Java Code Snippets**	**Code Snippet License**
vogella.com	70	626	Eclipse Public License 2.0
stacktips.com	154	296	restrictive; non-standard language (all copying disallowed)
androidtutorialpoint.com	82	622	licensing unclear
tutorialspoint.com	82	525	restrictive; non-standard language (only learning use permitted)
sitepoint.com	211	435	licensing rights remain with post creator
Total	*599*	*2,504*	

snippets (min = 1 LOC; max = 1,090 LOC; median = 9 LOC) with a creation date ranging from January, 2008 to January, 2018. As we already filtered the XML snippets in the tutorials, it was unnecessary to perform that step for the Stack Overflow snippet corpus.

Considering the large corpora of code snippets recovered from Stack Overflow and tutorials, in order to detect all the duplicate code snippet pairs, we applied our proposed *adaptive prefix filtering based scalable code clone detection tool*, able to rapidly process a large corpora of code [99]. For scalability, the code clone detection tool uses a textual representation of source code. We used a similarity threshold value of 0.8, i.e., detecting all the code clones that have at least 80% similar terms. The threshold value of 0.8 was used in similar studies in the past [90], as it retains the flexibility of detecting Type-1, Type-2 and Type-3 code clones while producing few false positives [78] [100]. As a means to further reduce possible false positives that could occur with small code snippets, we disregarded clone pairs where one of the snippets had fewer than 10 lines of code. Following this step, we observed 4,718 duplicate code pairs between the tutorial and Stack Overflow code snippets.

In examining the detected duplicate Android code snippets, we observed a high occurrence of clone pairs that represented standard Android generated (i.e. template) code. Clearly, these snippets were not copied from either source, but rather repre-

sented well-known patterns that the Android Studio IDE generates for a few common Android classes, e.g., Activity, Fragment. In order to filter these, we observed that the automatically generated (or template) code snippets tend to occur more frequently as code pairs between the two sources. Based on this observation, we selected a cut-off point of a maximum of 3 tutorial snippets that a single Stack Overflow snippet can map to. Larger numbers of tutorial matches were usually produced by templates. We used the course grained filtering as a guide, and examined the dataset manually to further detect template snippets. Our final set of duplicate code pairs between tutorials and Stack Overflow consists of 2,148 duplicate pairs, representing 346 unique tutorial snippets, and 1,488 unique Stack Overflow code snippets extracted from the SOTorrent dataset.

Developers copy code snippets due to various reasons. To identify the commonly occurring motives for code reuse between software development tutorial and Stack Overflow posts, two of the authors independently analyzed 100 randomly selected posts from Stack Overflow, including 50 questions and 50 answers, that were classified as code clones originating from tutorials. Based on the examined posts, each author devised a list of reasons (or categories) explaining why developers copied a code snippet and assigned one of them to each code clone. Differences between authors' annotation schemes were resolved via in-person discussion.

Due to the lack of reliable modification timestamps of the tutorials, we could not automatically discern which source contained the original code snippet and which source contained a copy. In many of our findings, where the source is unknown to us, we report on duplicate snippets. However, during the qualitative analysis of why snippets are copied, we were able to use contextual clues to relatively reliably predict where the snippet originated from. Such clues included links or references, existence of more context or code on one of the platforms, or notions of date or time.

3.7.2 Research Findings

In this section, we present the results of our analysis of duplicate code snippets between Stack Overflow posts and Android tutorials; we first describe developers' motives for copying code snippets followed by analysis of a few properties of the identified duplicates inluding evolution over time.

3.7.2.1 Understanding Code Snippets Copied from Tutorials to Stack Overflow

We used qualitative study techniques to build a taxonomy of rationales for code copying following the procedure described in Section 3.7.1. Based on the analysis of contextual information of the copied code blocks, the authors determined that *all of the randomly sampled snippets in the qualitative study were copied from tutorials to Stack Overflow*. We were unable to observe any snippets copied from Stack Overflow to tutorials, which could have been due to the specific parameters we used, e.g. minimum of 10 lines of code for a duplicate snippet. The final set of rationales for duplication, identified by two of the authors, is presented in Table 13. Note that depending on whether a Stack Overflow question or answer was examined, different non-overlapping sets of reasons for code reuse were discovered, hence we used questions and answers as the primary dimension in displaying the results.

The majority of cases when tutorial code snippets are copied to Stack Overflow's questions are related to experiencing an error or exception (28 out of 50 questions), or when the code does not work as intended by a developer (13 out of 50 questions). This result may be a consequence of many factors, such as e.g. errors in tutorials or a misconfigured IDE, although while analyzing questions' description, we often noted that developers tried to first modify the code from tutorials and once failed, they

Table 13.: Rationale for copying code snippets from tutorials to Stack Overflow posts.

Developer's Rationale	**Category**	**# Posts**	**Example Post**
Questions			
Facing exceptions or errors in the code	Error/Exception	28 / 50	*I'm trying to put data to my list view [...] using navigation drawer. I created a list view and defined the adapter but when I run it I got null pointer in the logcat. [...] [code snippet]*
Looking for help due to unexpected behavior	Unexpected behavior	13 / 50	*I have App1 and App2. App1 has the database in the content provider and App2 will insert data in database of App1, but when I call getContentResolver().insert(...), it always return null as uri. [code snippet] Please let me know the mistake, so I can solve it.*
Asking about implementation of a specific functionality	Functionality	7 / 50	*The problem is when I rotate the cellphone, the music starts again, how can I prevent that? [...][code snippet]*
Asking for help as a code snippet is not working with a particular version of Android API	Version compatibility	2 / 50	*I want to install a library to use PreferenceFragmentCompat or any class that replaces android.app.PreferenceFragment so my app can work in API 11 and lower. Can anyone please give me some details such as which library should I use and how to install it in my AS project? [code snippet]*
Answers			
Providing a solution/example implementing requested functionality	Example/Solution	48 / 50	*You need to override onSaveInstanceState(Bundle savedInstanceState) and write the application state values you want to change to the Bundle parameter like this [code snippet]*
Fixing errors in the code after developer modified code form tutorial	Fixing the code	1 / 50	*Here is the fixed setup, next time you need to do the imports for each object: [fixed code snippet]*
Providing additional information to support explanation	Clarification	1 / 50	[Question:] *Why should I use an additional layout file to present a ListView?* [Answer:] *When you are creating a simple ListView: [code snippet]. When creating a custom ListView: [code snippet]. In this example, if you look at the line: View rowView = inflater.inflate(R.layout.rowlayout, parent, false);, the R.layout.rowlayout is your custom layout used to show your custom ListView. Refer to the source link at the top of the answer for a detailed tutorial on ListView's.*

were reaching to Stack Overflow community asking for help and some clarification. Another rationale for copying code snippets is implementing a specific functionality (7 cases), which occurs when a developer wants to extend the code found in a tutorial, but has little knowledge on how to proceed. Additionally, we observed 2 questions that arose due to API compatibility issues between different Android API versions.

Providing an exemplary implementation for a specific issues was a prevalent motive for reusing code snippets from tutorials in Stack Overflow's answers. In 48 out of 50 answers we observed that developers copied the code to either directly resolve the question or to present a minimal working example. Additionally, we noted a singular case of an answer fixing the code provided in the question, where the code originated from the tutorial, indicating an unsuccessful attempt of modification. Finally, we also observed one answer, categorized as clarification, when developer used the copied code to provide an example supporting explanation to a posed question.

3.7.2.2 Properties and Evolution of Copied Code Snippets

We detected 2,148 duplicate snippets (346 snippets from tutorials and 1,488 snippets from Stack Overflow) originating from 189 tutorials and 1,398 Stack Overflow posts, including 909 questions and 489 answers.

To evaluate the popularity of reused code snippets, we studied the distribution of the number of up and down votes for Stack Overflow posts. Among 1,398 posts containing a duplicate code snippet, we found 637 up voted and 226 down voted posts. Note that a post can be both up and down voted on Stack Overflow. Figure 6 shows the distribution of up and down votes with respect to the number of posts. The majority of the up voted posts received between 1 and 4 votes, while about 64 posts gathered more than 5 up votes, including one post with over 2000 up votes. The similar shape of distribution is observed for the down votes, with a peak for the

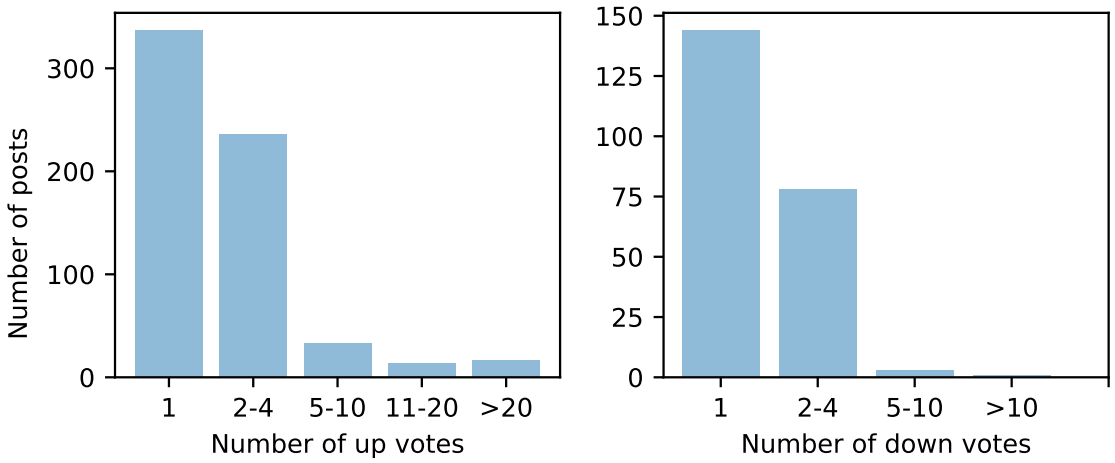

Fig. 6.: Popularity distribution of Stack Overflow posts containing code clones

number of votes between 1 and 4, however less than 10 posts were down voted more than 5 times. The relatively high number of up votes and the difference between the number of up voted posts when compared to the number of down votes indicates that code blocks copied from tutorials were considered as useful by the Stack Overflow's community. Moreover, we observed that 31% of answers containing copied code snippet were accepted as solutions to a question.

We used SOTorrent dataset of Stack Overflow post versions to examine the edit trends of the 1,488 duplicate code blocks. Note that a code block might not be edited in all versions of a Stack Overflow post, hence we distinguish between a case when two versions of a code block are the same (not-edited) or when they actually differ (edited). As a point of reference, we used an evolution analysis of Stack Overflow post blocks, including text and code blocks, presented by Baltes et al.[101]. Figure 7 shows distribution of the code duplicates' versions, considering both not-edited and edited cases. Among all the unique duplicate code snippets, 650 (43%) have more than one version, although only 256 of them (17%) have actually been modified. Most of the

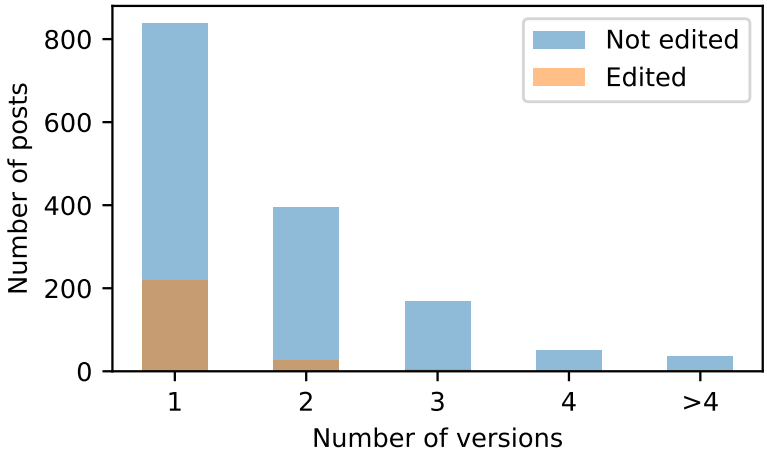

Fig. 7.: The number of versions of posts containing copied code snippets

edited code clones (86%) were modified only once and only 1% were modified more than three times. Baltes et al.[101] reported a similar result, with 46.6% edited post blocks. Although they did not provide separate analysis for the number of edited code blocks, they observed on average 4.1 code blocks versions, whereas majority of duplicate code snippets were edited only once. This may indicate that the reused code snippets are less likely to be updated as they originated from a trustworthy source, such as a tutorial.

To quantify the characteristics of modification, we measured the difference between the number of LOC and the number of characters comparing the first and the last version of a code snippet. We observed that on average 17.4 LOC (min = 1, max = 215, median = 5.5, std = 32.1) and 545.8 characters (min = 1, max = 8596, median = 145.5, std = 1119.1) in the code block were modified (either added or deleted).

We analyzed the timespan between edits of the reused code blocks with respect to the first and second time of the modification. The results are presented in Table 14. Overall, the edits characteristics of copied code snippets follows general trend observed for Stack Overflow posts as noted in [101], with first and second post edits occurring

Table 14.: Timespan of edits for the copied code snippets.

	Same year	**After 1 year**	**After 2 years**	**After 3 and years more**
First edits	230	6	8	3
Second edits	33	1	4	3

the same year post was created with probability of 90.3% and 88.3% respectively. In the case of copied code snippets, majority of the first edits (93%) take place in the same year as the post creation, while changing the code snippet the next year or later is rare, with respectively 2% and 5% of all cases. Similarly, the second edits occur most often during the year of posting the code (81%), and are less likely to be performed in the following year (2%) or later (17%). No copied code block was edited after five years of the creation. These results indicate that the developers tend to edit the reused code quickly, within a short period of time of posting the code snippets.

3.7.2.3 Threats to validity

Detection of code clones is potentially susceptible to several threats. One internal threat is related to configuration of the code clone detection tool and the heuristic used to filter false positives, as it directly affects the information we use for further analysis. To mitigate this threat, we followed similar studies to configure the code clone detection tool properly and examined false positive manually to find the most suited approach to remove false positives. Another threat arises from the fact that we did not explicitly check for duplication of code snippets within the tutorials or Stack Overflow posts, thus these may affect the total number of detected code clones. The results of the qualitative study pose an external threat since the observations were concluded over a limited number of reused code blocks.

3.8 Conclusions

Our proposed technique can outperform the most recent scalable code clone detection tool SourcererCC in terms of execution time within a certain range of similarity thresholds (between 40% to 80%). Since the technique directly extends SourcererCC, it produces the same output, duplicating the same high precision and recall as SourcererCC. We have evaluated our proposed code clone detection tool by randomly creating a subset of the IJaDataset 2.0 [79], a large code clone benchmark that contains 250MLOC and 25,000 open source Java systems.

Our experimental results indicate that our adaptive prefix filtering based code clone detection technique can be practically utilized in various code clone detection related applications that require large source code repository to be processed on a single machine. In our experiments we successfully performed code clone detection on a 17MLOC Java code base within a reasonable time window of several hours.

To the best of our knowledge our approach is also among few code clone detection techniques in the literature that can be additionally used for code clone search, which is the related problem of retrieving similar code blocks to a single code block issued as a query. Typically, code clone detection techniques make design decisions that allow them to only operate in batch mode, while code clone search requires the flexibility to answer numerous clone queries using a pre-built index. We show acceptable indexing and querying times for this application of our technique.

This dissertation reports on the quantity, type and evolution of duplicated source code snippets on Stack Overflow and Android tutorials. Our findings reveal some of the likely rationale of developers in copying code snippets from tutorials to Stack Overflow, the predominant direction of copying we encountered. Developers that reproduce code snippets from software tutorials do so to ask queries related to ob-

served errors or unexpected outputs or behavior. Developers sometimes use code snippets from tutorials to provide answers to Stack Overflow questions. The answers are marked as accepted on Stack Overflow with significant ratio (31%). Our findings also reveal that the duplicated code snippets between Stack Overflow and software tutorials can evolve over time, usually within the first year of the initial post creation.

3.9 Future Work

Our research also attempts to provide numerous examples and practical implementation advice for future applications of adaptive prefix filtering. As future work, we intend to better evaluate adaptive prefix filtering across a variety of languages and applications. We also aim to attempt to parallelize the technique to distributed memory architectures, which would greatly improve its scalability and extend its applicability .

CHAPTER 4

AUTOMATIC IDENTIFICATION OF VALID VERSION RANGE OF TUTORIALS

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 [102, 20]. They perform more than twenty software development related Web searches every day [103]. 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* [104].

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 [105]. 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 our research, we posit that one of the problems with modern Web tutorials, among other similar documentation types, is that they may not work with the de-

velopers' 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 [106] and 3.6 APIs per month are deprecated [107]. 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. [106].

Although researchers have proposed approaches aimed at extracting the mentions of API elements from Web resources [54], all of these techniques fail to manage the existence of numerous API versions with removed or deprecated API elements [50]. 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 [49]. 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.

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.

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 [5, 108, 109]. 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 [5]. 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 [110, 111, 112]. For instance, researchers have examined why APIs change [113] and how API changes affect developers [114]. Researchers studying whether changes in APIs trigger StackOverflow questions detected a strong increase in the number

of questions asked about frequently changed API methods [115]. 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 [24]. 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 8 shows a tutorial [116] 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 9, 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.

4.1 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 15 shows the release dates of each Android version. New releases are frequent and API changes, including

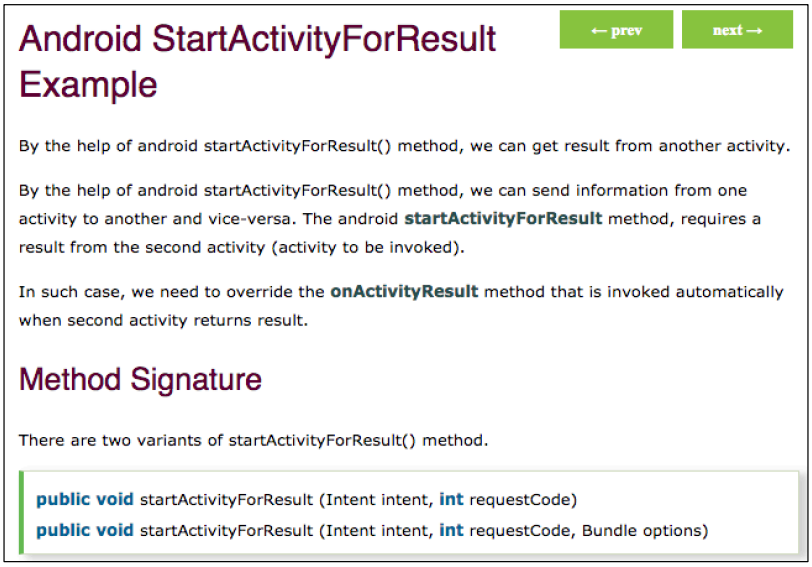

Fig. 8.: Portion of a tutorial discussing the `startActivityForResult` Android API [116]

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 10. 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 15 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

Fig. 9.: Portion of a tutorial utilizing the deprecated API `android.util.FloatMath` [15].

classes were removed. We used the listing of Android APIs provided by the Android Official Documentation [117], Android API Differences Report [118], and Android Support Library API Differences Report [119].

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 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

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 15.: The set of different Android versions, corresponding API levels, and number of removed and deprecated classes. [120] [117] [118]

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, fields, packages) by parsing the Android Official Documentation [117]. 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.

4.1.1 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

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 16.: Range of valid versions of the tutorials in our set.

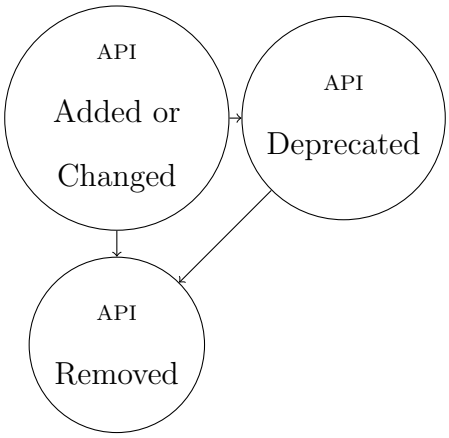

Fig. 10.: Lifecycle of an API element.

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.

4.1.2 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 16. 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 16 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.

4.1.3 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.

4.2 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 our research, 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.

4.2.1 Versioning Workflow

Figure 11 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 [117], Android API Differences Report [118], and the Android Support Library API Differences Report [119] 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 [117]. 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 [118], and Android Support Library API Differences Report [119].

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

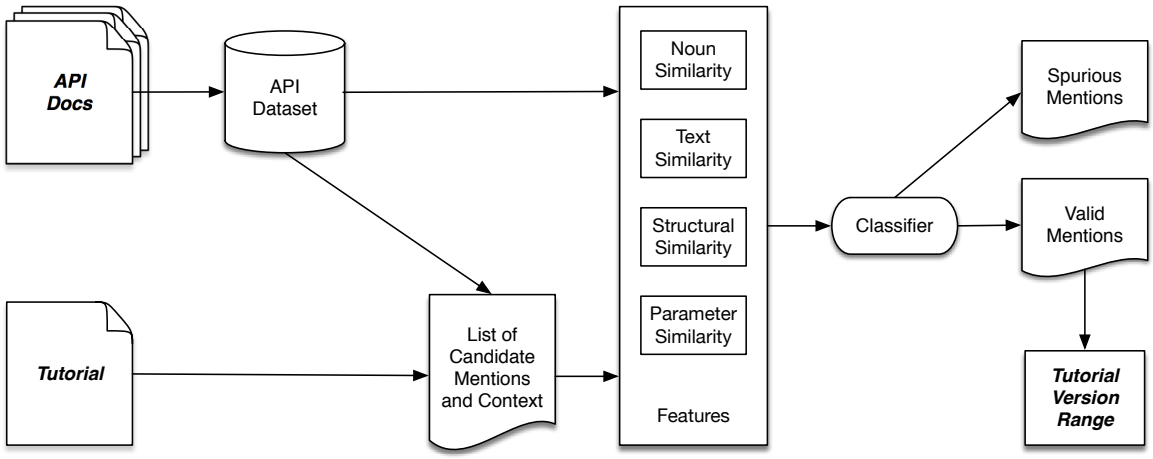

Fig. 11.: Overview of our technique for automated versioning of software development tutorials.

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.

4.2.2 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 re-

lated 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 8 – to illustrate each feature.

4.2.2.1 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 [54]. 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 8 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 occur-

ring in the description of this method in the API documentation, e.g., `[['Intent', 'int', 'Bundle', 'options']`.

4.2.2.2 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 8 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 8) 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."*

4.2.2.3 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 8, 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. [54], 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 8 has no

source code in its vicinity, this Structural Similarity will be zero.

4.2.2.4 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 8 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:

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

,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 8, its Parameter Similarity is also zero.

4.3 Experimental Analysis

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**.

4.3.1 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 [121] and TextBlob [122]. For island parsing of the code snippets embedded in tutorials we used the SrcML parser [123].

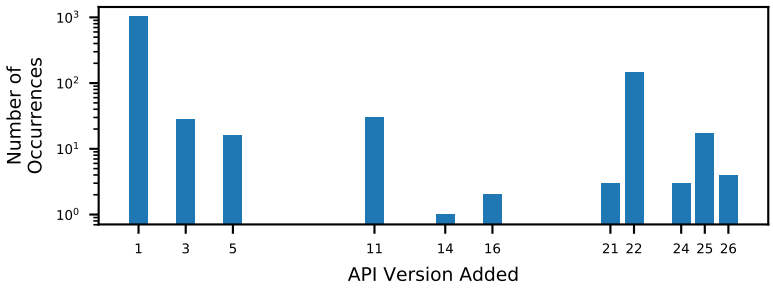

Fig. 12.: The distribution of the added versions across all the tutorial (log scale).

The versions when API mentions in our tutorial dataset were initially introduced in Android follow the distribution shown in Figure 12. 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.

- *Precision* – measures the ability of a classifier in labeling positive samples as positive and avoid labelling positive samples as negative [124]. 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 positives is high. [125]. 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 [124]. 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 [125]. 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 [126].

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 [127]. 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 [128]. 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 [129], 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 documenta-

tion. 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) [130], with its default settings (i.e. *kernel=linear and maximum iterations=5000).*

4.3.2 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 18 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%.

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 17. 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 19.

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%.

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 8.

The results for **RQ2** are shown in Table 20. 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 our research. 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.

4.3.3 Additional Feature Based on Word Embeddings

To improve the precision and recall of our tutorial versioning technique, we added an additional feature that computes similarity based on word embeddings. By using a word embeddings model trained on a corpus of Stack Overflow posts, SO_word2vec [131], we can semantically extend both the official API description and context of potential mentions of the tutorials and then calculate the cosine similarity

between them. Table 21 shows the evaluation results with this additional feature when using tutorials as unit. We observe that using word embeddings, the precision and recall have improved significantly on the combined dataset with 61% precision, 55% recall and 55% F1-score and using the Random Forest Classifier (*numTrees= 1500*).

Table 22 shows the overall tutorial versioning effectiveness with the additional word embeddings features. The results show that although the new feature increases precision, recall and F1 score, the average distance of predicted version range is slightly larger (5.84) than the multi-instance classifier using the base set of features. The reason for this discrepancy is just that the misclassified APIs using the new features happen to introduce a higher penalty to the overall versioning accuracy.

4.4 Versioning Video Tutorials

Online video tutorials are also a popular source of information for software developers. Like conventional text-based tutorials, video tutorials provide step-by-step information for a particular development topic. Video tutorials target developers that can learn quickly with visual and auditory cognition [132]. Video is preferred by some developers as they provide deeper context, more examples and the ability to observe all changes made to the source code. By observing the IDE and OS environment, developers can clearly relate the output to the source code [105].

In the following, we have extended our research study to exampine our versioning technique's applicability towards online video tutorials. We again focus our experimentation on Android (video) tutorials.

As before, we aim to select a set of tutorials for experimentation using stratified sampling. For this purpose we gathered a corpus of Android tutorials freely available on the Web and extracted their metadata including information related to the video

tutorial's *Title*, *Publish Date*, *Number of Views*, and *Number of Likes*. We applied stratified sampling based only on the *Publish Date* and *Number of Views* randomly selecting 7 tutorials for annotation, one per category. After manually annotating these 7 tutorials, our evalation dataset contains approximately 40,000 (Mention,Candidate) pairs.

Before annotating the tutorials, we preprocessed them to extract the textual representation of their contents (textual representations of sound and images) using the following set of steps.

- We apply the Google2Sr [134] tool to generate text files of video transcript. There are 7 transcripts for each of the 7 tutorials.
- We apply the FFMPEG [135] tool to extract video frames. The video frames of tutorials have been extracted every 10 seconds. From our 7 tutorials, we extracted 302 video frames. Most of these frames contain repetetive content. We have manually filtered the video frames that contain unique information, keeping a set of 22 video frames that contain different information from each other.
- After extracting video frames we apply the TESSERACT-OCR [136] tool to extract text files that contain source code and natural language text/description from the images.
- Finally we annotate these text files to extract API mentions.

To determine the versions of each API mentioned in the video tutorials, we train a Random Forest classifier, and evaluate with 7-fold cross validation. We experiment with different types of API elements (i.e. method, class, field) or a combination of all three. Table 23 shows the results of this experiement. Our technique exhibits a

high precision (81%), recall (81%), and F1-score (81%) for a combination of all API elements, performing best on fields, followed by classes, with methods performing the lowest, with an F1-score of 71%.

We also evaluate our technique using tutorials as units, where we use mentions of a specific API type from 6 of the 7 tutorials as a training set and use the final tutorial as the test set. The results are shown in Table 24. For this configuration we observe lower results than when using all mentions without regard for tutorial boundaries. Although precision and recall are moderate for classes and fields, we observe extremely poor values for method level (average F1-score of 9%) and combined (average F1-score of 32%) dataset. Considering the implication of API classification to the overall tutorial versioning, we observe that the average distance of true and predicted version range is very large, around 13, as shown in Table 25.

We argue that the low precision, recall of our technique when using tutorial as unit implies the difficulty of extracting the information (source code and text context) of video tutorials rather than the applicability of our proposed approach. By examining the results, we construct two main reasons why our methodology is giving low precision and recall.

- *Lack of natural language context:* Video tutorials do not have the advantage of in-depth explanation like text tutorials. Therefore, they lack sufficient natural language descriptions. Usually, video tutorials provide information to the learners with visual and auditory cues [132], therefore, a detailed textual explanation does not exist in the video. As a result, our technique fails to extract meaningful textual features from many video tutorials.

- *Difficulty in extracting source code and descriptions:* Text extracted from video tutorials is extremely noisy. We have used an OCR tool on the video frames,

however, the output of such tools can be succeptible to noise [137]. A high amount of noise can occur because the background of the video frames is sometimes difficult to process and because the quality of the video frames can also be very low [105]. When code snippets used in the video tutorials are interspersed with application execution then it is very difficult to derive structural features and parameter features [132]. In some video tutorials, code is written incrementally while the author provides step by step explanations on each line. Some explanations cause the author to quickly jump through the code base, showing dissimilar portions of code in different video frames. In these cases, it is difficult to extract relevat information related to API mentions from each video frame. [105].

4.5 Conclusions and Future Work

In this chapter, we proposed a novel idea of inferring the version of informal software documentation avaiable on the Web, focusing specifically on textual and video tutorials available on the Web. 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.

To improve the results of versioning video tutorials specifically, we plan to apply additional techniques to reduce noise of videos and correctly extract source code and textual description. This is an area where new tools, based on latest artificial intelligence techniques, are continuously becoming available. We aim to experiement with cutting edge tools, as improvements with converting videos to text should lead to strong downstream improvements in our ability to version these types of tutorials.

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 17.: Results using tutorials as units. (-These tutorials have no positive fields.)

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

Table 18.: 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 19.: Results of multi instance classification.

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 20.: Comparison of the true and predicted version ranges of tutorials.

API Elements	Precision	Recall	F1 score
Class	74%	79%	74%
Field	63%	52%	53%
Method	61%	41%	44%
Combined	61%	55%	55%

Table 21.: Results of using tutorials as units with the additional word embedding-based feature.

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]	[24-27]	2
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]	[25-28]	3
Android Navigation Drawer – for Sliding Menu / Sidebar	[25-28]	[24-28]	1
Building Android applications with Gradle - Tutorial	[1-28]	[1-21]	7
Android Facebook Login Tutorial – Integrating Facebook SDK 4	[26-28]	[24-27]	3
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]	[3-28]	23
Android Chat Bubble Layout – with 9 patch Image using ListView	[25-28]	[26-28]	0
Understanding Androids Parcelable - Tutorial	[1-28]	[1-28]	0
Average Distance			5.84

Table 22.: Comparison of the true and predicted version ranges of tutorials with the additional word embedding-based feature.

API Elements	Precision	Recall	F1 score
Class	85%	85%	85%
Field	93%	93%	93%
Method	71%	71%	71%
Combined	81%	81%	81%

Table 23.: Results of Video tutorials using mentions as units.

API Elements	Precision	Recall	F1 score
Class	46%	61%	47%
Field	60%	55%	56%
Method	12%	7%	9%
Combined	40%	33%	32%

Table 24.: Results of Video tutorials using tutorials as units.

Tutorial Name	Version Range	Predicted Version Range	Manhattan Distance
Generating Random Number	[24-28]	[1-28]	23
Android ListView With Image	[21-28]	[1-28]	20
Android Intents Example	[1-28]	[28-28]	27
Android Intents Send Image from Drawable Folder	[19-28]	[24-28]	5
Androidtutorial Enable Up Navigation for your Android application	[1-28]	[1-28]	0
Create HelloWorld Android application	[1-28]	[1-28]	0
Introduction to Android Notifications	[17-21]	[1-22]	17
Average Distance			13.14

Table 25.: Comparison of the true and predicted version ranges of Video tutorials.

CHAPTER 5

FINAL CONCLUSIONS

Software development tutorials, available in abundance on the Web, are very popular among software developers due to their applicability to a variety of software development problems. This dissertation presents a novel approach of assessing the quality of software development tutorials. While quality is subjective, we defined in terms of several practical scenarios where tutorials can fail developer's expectations: outdated or incompatible software versions, license conflict and security breaches in the tutorial's code snippets. Outdated tutorials waste development time as the developer takes time to realize the incompatibility exist. Code snippets that carry strict licenses or software vulnerabilities present an even more serious problem, as developers reusing tutorial contents can propagate these issues into their project.

To address these problems, this dissertation describes two techniques: *Web-scale textual code clone search* that can detect duplicate code snippets with reasonable speed and scalability in order to detect their origin; and *automatic identification of valid version ranges* that can use the official API documentation as a trusted source to detect a tutorial's version.

For the automatic identification of valid version of API we experiment with tutorials for the popular Android platform. In Android, existing APIs are often removed or deprecated and new APIs are added to keep pace with new tools and technologies. The removed or deprecated APIs, when used in source code snippets or in the textual description of the tutorials, make the tutorials not compatible with the most updated versions. Our approach to extract the version compatibility of the

software development tutorials uses a two step process. As the first step, we extract all the API mentions of the tutorials differentiating API mentions from similar verbs or nouns. As the second step, we disambiguate the overloaded APIs. Our results show that disambiguation of method level APIs is more challenging than class or field granularity, due to the high number of synonymous API methods lacking substantial description both in the official API documentation and the surrounding texts of the code snippets of tutorials. However, despite this difficulty of determining versions of some method level APIs, we have effectively extracted versions at other granularity levels such as classes and fields, so, overall, our proposed approach can usually predict the overall tutorial version range with reasonable accuracy.

The dissertation also describes an application of our scalable code clone detection technique for the empirical study of properties and characterization of duplicate code snippets between Stack Overflow and tutorials. Our qualitative analysis reveals that developers often copy code snippets between Stack Overflow and Android tutorials in order to gain insights about copied code snippets. They often need help as the copied code snippets may not be functional as there exists errors or unexpected outcomes during execution or developers need clarification about the copied code snippets. Moreover they need help to implement specific functionality. Code snippets are also usually copied in the Stack Overflow as solution to a particular problems.

REFERENCES

[1] Ivan Srba and Maria Bielikova. "A Comprehensive Survey and Classification of Approaches for Community Question Answering". In: *ACM Trans. Web* 10.3 (Aug. 2016), 18:1–18:63. ISSN: 1559-1131. DOI: 10.1145/2934687. URL: http://doi.acm.org/10.1145/2934687.

[2] Jacob Krüger et al. "Empirical Studies in Question-answering Systems: A Discussion". In: *Proceedings of the 5th International Workshop on Conducting Empirical Studies in Industry*. CESI '17. Buenos Aires, Argentina: IEEE Press, 2017, pp. 23–26. ISBN: 978-1-5386-1546-1. DOI: 10.1109/CESI.2017.6. URL: https://doi.org/10.1109/CESI.2017.6.

[3] C. H. Kao. "Collaboration framework for software development based on question and answer sites". In: *2018 IEEE International Conference on Applied System Invention (ICASI)*. 2018, pp. 310–313. DOI: 10.1109/ICASI.2018.8394595.

[4] Pradeep Kumar Roy et al. "Finding and Ranking High-Quality Answers in Community Question Answering Sites". In: *Global Journal of Flexible Systems Management* 19.1 (2018), pp. 53–68. ISSN: 0974-0198. DOI: 10.1007/s40171-017-0172-6. URL: https://doi.org/10.1007/s40171-017-0172-6.

[5] Rebecca Tiarks and Walid Maalej. "How Does a Typical Tutorial for Mobile Development Look Like?" In: *Proceedings of the 11th Working Conference on Mining Software Repositories*. MSR 2014. Hyderabad, India: ACM, 2014, pp. 272–281. ISBN: 978-1-4503-2863-0. DOI: 10.1145/2597073.2597106. URL: http://doi.acm.org/10.1145/2597073.2597106.

[6] Keith Burghardt et al. “The myopia of crowds: Cognitive load and collective evaluation of answers on Stack Exchange”. In: *PLoS ONE* 12.3 (2017).

[7] I. Srba and M. Bielikova. “Why is Stack Overflow Failing? Preserving Sustainability in Community Question Answering”. In: *IEEE Software* 33.4 (2016), pp. 80–89. ISSN: 0740-7459. DOI: 10.1109/MS.2016.34.

[8] Mario Linares-Vásquez. “Supporting Evolution and Maintenance of Android Apps”. In: *Companion Proceedings of the 36th International Conference on Software Engineering*. ICSE Companion 2014. Hyderabad, India: ACM, 2014, pp. 714–717. ISBN: 978-1-4503-2768-8. DOI: 10.1145/2591062.2591092. URL: http://doi.acm.org/10.1145/2591062.2591092.

[9] Tianyi Zhang et al. “Are Code Examples on an Online Q&A Forum Reliable?: A Study of API Misuse on Stack Overflow”. In: *Proceedings of the 40th International Conference on Software Engineering*. ICSE ’18. Gothenburg, Sweden: ACM, 2018, pp. 886–896. ISBN: 978-1-4503-5638-1. DOI: 10.1145/3180155.3180260. URL: http://doi.acm.org/10.1145/3180155.3180260.

[10] Chaiyong Ragkhitwetsagul et al. “Toxic Code Snippets on Stack Overflow”. In: *arXiv preprint arXiv:1806.07659* (2018).

[11] Y. Acar et al. “You Get Where You’re Looking for: The Impact of Information Sources on Code Security”. In: *2016 IEEE Symposium on Security and Privacy (SP)*. 2016, pp. 289–305. DOI: 10.1109/SP.2016.25.

[12] Felix Fischer et al. “Stack overflow considered harmful? the impact of copy&paste on android application security”. In: *Security and Privacy (SP), 2017 IEEE Symposium on*. IEEE. 2017, pp. 121–136.

[13] Le An et al. “Stack overflow: a code laundering platform?” In: *Software Analysis, Evolution and Reengineering (SANER), 2017 IEEE 24th International Conference on*. IEEE. 2017, pp. 283–293.

[14] Sebastian Baltes and Stephan Diehl. “Usage and Attribution of Stack Overflow Code Snippets in GitHub Projects”. In: *arXiv preprint arXiv:1802.02938* (2018).

[15] JASON MCREYNOLDS. *Android Tutorial: Implement A Shake Listener*. `http://jasonmcreynolds.com/?p=388`. 2013.

[16] *Stack Overflow Code Snippets*. `https://stackoverflow.com/questions/15577307/how-to-extend-sqlitedatabase-class/15577409#15577409`. March, 2013.

[17] *Tutorial Code Snippets*. `https://www.vogella.com/tutorials/AndroidSQLite/article.html`. July, 2017.

[18] M. Allahbakhsh et al. “Quality Control in Crowdsourcing Systems: Issues and Directions”. In: *IEEE Internet Computing* 17.2 (2013), pp. 76–81. ISSN: 1089-7801. DOI: 10.1109/MIC.2013.20.

[19] Yuhao Wu et al. “How do developers utilize source code from stack overflow?” In: *Empirical Software Engineering* (2018). ISSN: 1573-7616. DOI: 10.1007/s10664-018-9634-5. URL: https://doi.org/10.1007/s10664-018-9634-5.

[20] Preetha Chatterjee et al. “What information about code snippets is available in different software-related documents? an exploratory study”. In: *Software Analysis, Evolution and Reengineering (SANER), 2017 IEEE 24th International Conference on*. SANER 2017. Klagenfurt, Austria: IEEE, 2017,

pp. 382–386. ISBN: 978-1-5090-5501-2. DOI: 10.1109/SANER.2017.7884638. URL: https://ieeexplore.ieee.org/document/7884638.

[21] *software-development-resources*. https://www.qasymphony.com/blog/101-software-development-resources/. 2018.

[22] MohammadReza Tavakoli, Abbas Heydarnoori, and Mohammad Ghafari. "Improving the Quality of Code Snippets in Stack Overflow". In: *Proceedings of the 31st Annual ACM Symposium on Applied Computing*. SAC '16. Pisa, Italy: ACM, 2016, pp. 1492–1497. ISBN: 978-1-4503-3739-7. DOI: 10.1145/2851613.2851789. URL: http://doi.acm.org/10.1145/2851613.2851789.

[23] Hapnes Toba et al. "Discovering high quality answers in community question answering archives using a hierarchy of classifiers". In: *Information Sciences* 261 (2014), pp. 101–115.

[24] David Kavaler and Vladimir Filkov. "Determinants of quality, latency, and amount of Stack Overflow answers about recent Android APIs". In: *PloS one*. Vol. 13. 3. Bremen, Germany: Public Library of Science, 2018, e0194139. DOI: 10.1371/journal.pone.0194139. URL: https://doi.org/10.1371/journal.pone.0194139.

[25] Antoaneta Baltadzhieva and Grzegorz Chrupała. "Question quality in community question answering forums: a survey". In: *Acm Sigkdd Explorations Newsletter* 17.1 (2015), pp. 8–13.

[26] Muhammad Asaduzzaman et al. "Answering Questions About Unanswered Questions of Stack Overflow". In: *Proceedings of the 10th Working Conference on Mining Software Repositories*. MSR '13. San Francisco, CA, USA: IEEE Press, 2013, pp. 97–100. ISBN: 978-1-4673-2936-1. URL: http://dl.acm.org/citation.cfm?id=2487085.2487109.

[27] Chaiyong Ragkhitwetsagul, Jens Krinke, and Rocco Oliveto. "Awareness and Experience of Developers to Outdated and License-Violating Code on Stack Overflow: An Online Survey". In: *arXiv preprint arXiv:1806.08149* (2018).

[28] D. M. German et al. "Code siblings: Technical and legal implications of copying code between applications". In: *2009 6th IEEE International Working Conference on Mining Software Repositories*. 2009, pp. 81–90. DOI: 10.1109/MSR.2009.5069483.

[29] Yanfang Ye et al. "ICSD: An Automatic System for Insecure Code Snippet Detection in Stack Overflow over Heterogeneous Information Network". In: (2018).

[30] Mu-Woong Lee et al. "Instant Code Clone Search". In: *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering*. FSE '10. Santa Fe, New Mexico, USA: ACM, 2010, pp. 167–176. ISBN: 978-1-60558-791-2.

[31] Iman Keivanloo, Juergen Rilling, and Philippe Charland. "Internet-scale real-time code clone search via multi-level indexing". In: *Reverse Engineering (WCRE), 2011 18th Working Conference on*. IEEE. 2011, pp. 23–27.

[32] Iman Keivanloo, Juergen Rilling, and Philippe Charland. "Seclone-a hybrid approach to internet-scale real-time code clone search". In: *Program Comprehension (ICPC), 2011 IEEE 19th International Conference on*. IEEE. 2011, pp. 223–224.

[33] Iman Keivanloo, Juergen Rilling, and Ying Zou. "Spotting Working Code Examples". In: *Proceedings of the 36th International Conference on Software Engineering*. ICSE 2014. Hyderabad, India: ACM, 2014, pp. 664–675. ISBN: 978-1-4503-2756-5.

[34] Brenda S Baker. "A theory of parameterized pattern matching: algorithms and applications". In: *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*. ACM. 1993, pp. 71–80.

[35] Rainer Koschke. "Large-scale inter-system clone detection using suffix trees and hashing". In: *Journal of Software: Evolution and Process* 26.8 (2014), pp. 747–769.

[36] Toshihiro Kamiya, Shinji Kusumoto, and Katsuro Inoue. "CCFinder: a multilinguistic token-based code clone detection system for large scale source code". In: *IEEE Transactions on Software Engineering* 28.7 (2002), pp. 654–670.

[37] Benjamin Hummel et al. "Index-based code clone detection: incremental, distributed, scalable". In: *Software Maintenance (ICSM), 2010 IEEE International Conference on*. IEEE. 2010, pp. 1–9.

[38] Lingxiao Jiang et al. "DECKARD: Scalable and Accurate Tree-Based Detection of Code Clones". In: *Proceedings of the 29th International Conference on Software Engineering*. ICSE '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 96–105. ISBN: 0-7695-2828-7.

[39] James R Cordy and Chanchal K Roy. "The NiCad clone detector". In: *Program Comprehension (ICPC), 2011 IEEE 19th International Conference on*. IEEE. 2011, pp. 219–220.

[40] Hitesh Sajnani et al. "SourcererCC: Scaling Code Clone Detection to Bigcode". In: *Proceedings of the 38th International Conference on Software Engineering*. ICSE '16. Austin, Texas: ACM, 2016, pp. 1157–1168. ISBN: 978-1-4503-3900-1.

[41] Nils Göde and Rainer Koschke. "Incremental clone detection". In: *Software Maintenance and Reengineering, 2009. CSMR'09. 13th European Conference on*. IEEE. 2009, pp. 219–228.

[42] Simone Livieri et al. "Very-Large Scale Code Clone Analysis and Visualization of Open Source Programs Using Distributed CCFinder: D-CCFinder". In: *Proceedings of the 29th International Conference on Software Engineering*. ICSE '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 106–115. ISBN: 0-7695-2828-7.

[43] Jeffrey Svajlenko, Iman Keivanloo, and Chanchal K Roy. "Scaling classical clone detection tools for ultra-large datasets: An exploratory study". In: *Proceedings of the 7th International Workshop on Software Clones*. IEEE Press. 2013, pp. 16–22.

[44] Sunita Sarawagi and Alok Kirpal. "Efficient set joins on similarity predicates". In: *Proceedings of the 2004 ACM SIGMOD international conference on Management of data*. ACM. 2004, pp. 743–754.

[45] Rares Vernica, Michael J Carey, and Chen Li. "Efficient parallel set-similarity joins using MapReduce". In: *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*. ACM. 2010, pp. 495–506.

[46] Seulbae Kim et al. "VUDDY: A Scalable Approach for Vulnerable Code Clone Discovery". In: *Security and Privacy (SP), 2017 IEEE Symposium on. IEEE*. 2017.

[47] Jiannan Wang, Guoliang Li, and Jianhua Feng. "Can we beat the prefix filtering?: an adaptive framework for similarity join and search". In: *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. ACM. 2012, pp. 85–96.

[48] Aline Brito et al. “APIDiff: Detecting API breaking changes”. In: *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*. SANER ’18. Campobasso, Italy: IEEE Computer Society, 2018, pp. 507–511. ISBN: 978-1-5386-4970-1. DOI: 10.1109/SANER.2018.8330249. URL: http://doi.ieeecomputersociety.org/10.1109/SANER.2018.8330249.

[49] Tianyue Luo et al. “MAD-API: Detection, Correction and Explanation of API Misuses in Distributed Android Applications”. In: *Artificial Intelligence and Mobile Services – AIMS 2018*. Cham: Springer International Publishing, 2018, pp. 123–140. ISBN: 978-3-319-94361-9. DOI: 10.1007/978-3-319-94361-9_10. URL: https://link.springer.com/chapter/10.1007\%2F978-3-319-94361-9_10.

[50] Jing Zhou and Robert J. Walker. “API Deprecation: A Retrospective Analysis and Detection Method for Code Examples on the Web”. In: *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. FSE 2016. Seattle, WA, USA: ACM, 2016, pp. 266–277. ISBN: 978-1-4503-4218-6. DOI: 10.1145/2950290.2950298. URL: http://doi.acm.org/10.1145/2950290.2950298.

[51] SM Nasehi et al. “What makes a good code example?: A study of programming Q&A in StackOverflow”. In: *Software Maintenance (ICSM), 2012 28th IEEE International Conference on*. ICSM 2012. Trento, Italy: IEEE, 2012, pp. 25–34. ISBN: 978-1-4673-2312-3. DOI: 10.1109/ICSM.2012.6405249. URL: https://ieeexplore.ieee.org/document/6405249.

[52] Deheng Ye et al. “Learning to extract api mentions from informal natural language discussions”. In: *Software Maintenance and Evolution (ICSME), 2016*

IEEE International Conference on. ICSME 2016. Raleigh, NC, USA: IEEE, 2016, pp. 389–399. ISBN: 978-1-5090-3806-0. DOI: 10.1109/ICSME.2016.11. URL: https://ieeexplore.ieee.org/document/7816484.

[53] Deheng Ye et al. "APIReal: an API recognition and linking approach for online developer forums". In: *Empirical Software Engineering*. New York, NY, USA: Springer, 2018, pp. 1–32. DOI: 10.1007/s10664-018-9608-7. URL: https://doi.org/10.1007/s10664-018-9608-7.

[54] Gias Uddin and Martin P. Robillard. "Resolving API Mentions in Informal Documents". In: *CoRR* abs/1709.02396 (2017). arXiv: 1709.02396. URL: http://arxiv.org/abs/1709.02396.

[55] Barthélémy Dagenais and Martin P. Robillard. "Recovering Traceability Links Between an API and Its Learning Resources". In: *Proceedings of the 34th International Conference on Software Engineering*. ICSE '12. Zurich, Switzerland: IEEE Press, 2012, pp. 47–57. ISBN: 978-1-4673-1067-3. DOI: 10.1109/ICSE.2012.6227207. URL: http://dl.acm.org/citation.cfm?id=2337223.2337230.

[56] Peter C. Rigby and Martin P. Robillard. "Discovering Essential Code Elements in Informal Documentation". In: *Proceedings of the 2013 International Conference on Software Engineering*. ICSE '13. San Francisco, CA, USA: IEEE Press, 2013, pp. 832–841. ISBN: 978-1-4673-3076-3. DOI: 10.1109/ICSE.2013.6606629. URL: http://dl.acm.org/citation.cfm?id=2486788.2486897.

[57] Gayane Petrosyan, Martin P. Robillard, and Renato De Mori. "Discovering Information Explaining API Types Using Text Classification". In: *Proceedings of the 37th International Conference on Software Engineering - Volume 1*.

ICSE '15. Florence, Italy: IEEE Press, 2015, pp. 869–879. ISBN: 978-1-4799-1934-5. DOI: DOI:10.1109/ICSE.2015.97. URL: http://dl.acm.org/citation.cfm?id=2818754.2818859.

[58] He Jiang et al. "A more accurate model for finding tutorial segments explaining APIs". In: *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. Vol. 1. IEEE. 2016, pp. 157–167.

[59] He Jiang et al. "An Unsupervised Approach for Discovering Relevant Tutorial Fragments for APIs". In: *Proceedings of the 39th International Conference on Software Engineering*. ICSE '17. Buenos Aires, Argentina: IEEE Press, 2017, pp. 38–48. ISBN: 978-1-5386-3868-2. DOI: 10.1109/ICSE.2017.12. URL: https://doi.org/10.1109/ICSE.2017.12.

[60] Di Wu et al. "Automatically Answering API-related Questions". In: *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*. ICSE '18. Gothenburg, Sweden: ACM, 2018, pp. 270–271. ISBN: 978-1-4503-5663-3. DOI: 10.1145/3183440.3194965. URL: http://doi.acm.org/10.1145/3183440.3194965.

[61] Jingxuan Zhang et al. "Recommending APIs for API Related Questions in Stack Overflow". In: *IEEE Access*. IEEE '13. Piscataway, NJ, USA: IEEE, 2013, pp. 6205–6219. DOI: 10.1109/ACCESS.2017.2777845. URL: https://ieeexplore.ieee.org/abstract/document/8121986.

[62] Mohammad Masudur Rahman, Chanchal K Roy, and David Lo. "Rack: Automatic api recommendation using crowdsourced knowledge". In: *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. SANER '16. Suita, Japan: IEEE Press, 2016, pp. 349–

359. ISBN: 978-1-5090-1855-0. DOI: 10.1109/SANER.2016.80. URL: https://ieeexplore.ieee.org/document/7476656.

[63] Chanchal Kumar Roy and James R Cordy. "A survey on software clone detection research". In: *Queen's School of Computing TR* 541.115 (2007), pp. 64–68.

[64] Ira D Baxter et al. "Clone detection using abstract syntax trees". In: *Software Maintenance, 1998. Proceedings., International Conference on*. IEEE. 1998, pp. 368–377.

[65] Debarshi Chatterji et al. "Effects of cloned code on software maintainability: A replicated developer study". In: *Reverse Engineering (WCRE), 2013 20th Working Conference on*. IEEE. 2013, pp. 112–121.

[66] Debarshi Chatterji et al. "Measuring the efficacy of code clone information in a bug localization task: An empirical study". In: *Empirical Software Engineering and Measurement (ESEM), 2011 International Symposium on*. IEEE. 2011, pp. 20–29.

[67] Chanchal K Roy and James R Cordy. "An empirical study of function clones in open source software". In: *Reverse Engineering, 2008. WCRE'08. 15th Working Conference on*. IEEE. 2008, pp. 81–90.

[68] Brenda S Baker. "On finding duplication and near-duplication in large software systems". In: *Reverse Engineering, 1995., Proceedings of 2nd Working Conference on*. IEEE. 1995, pp. 86–95.

[69] Tomoya Ishihara et al. "Inter-project functional clone detection toward building libraries-an empirical study on 13,000 projects". In: *Reverse Engineering (WCRE), 2012 19th Working Conference on*. IEEE. 2012, pp. 387–391.

[70] Rainer Koschke. “Large-scale inter-system clone detection using suffix trees”. In: *Software Maintenance and Reengineering (CSMR), 2012 16th European Conference on*. IEEE. 2012, pp. 309–318.

[71] Daniel M German et al. “Code siblings: Technical and legal implications of copying code between applications”. In: *Mining Software Repositories, 2009. MSR'09. 6th IEEE International Working Conference on*. IEEE. 2009, pp. 81–90.

[72] Armijn Hemel and Rainer Koschke. “Reverse engineering variability in source code using clone detection: A case study for linux variants of consumer electronic devices”. In: *Reverse Engineering (WCRE), 2012 19th Working Conference on*. IEEE. 2012, pp. 357–366.

[73] Julius Davies et al. “Software bertillonage: finding the provenance of an entity”. In: *Proceedings of the 8th working conference on mining software repositories*. ACM. 2011, pp. 183–192.

[74] Takanobu Yamashina et al. “SHINOBI: A real-time code clone detection tool for software maintenance”. In: *Nara Institute of Science and Technology* (2008), p. 26.

[75] Kai Chen, Peng Liu, and Yingjun Zhang. “Achieving accuracy and scalability simultaneously in detecting application clones on android markets”. In: *Proceedings of the 36th International Conference on Software Engineering*. ACM. 2014, pp. 175–186.

[76] Chanchal K Roy, Minhaz F Zibran, and Rainer Koschke. “The vision of software clone management: Past, present, and future (keynote paper)”. In: *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on*. IEEE. 2014, pp. 18–33.

[77] Chanchal K Roy and James R Cordy. "Near-miss function clones in open source software: an empirical study". In: *Journal of Software: Evolution and Process* 22.3 (2010), pp. 165–189.

[78] Jeffrey Svajlenko et al. "Towards a big data curated benchmark of inter-project code clones". In: *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*. IEEE. 2014, pp. 476–480.

[79] *Ambient Software Evoluton Group. IJaDataset 2.0.* `http://secold.org/projects/seclone`. June 2017.

[80] Abdullah Sheneamer and Jugal Kalita. "A Survey of Software Clone Detection Techniques". In: *International Journal of Computer Applications* (2016), pp. 0975–8887.

[81] *Apache Lucene*. `https://lucene.apache.org/core/`. June 2017.

[82] Jeffrey Svajlenko, Iman Keivanloo, and Chanchal K Roy. "Big data clone detection using classical detectors: an exploratory study". In: *Journal of Software: Evolution and Process* 27.6 (2015), pp. 430–464.

[83] Jeffrey Svajlenko and Chanchal K Roy. "Evaluating clone detection tools with bigclonebench". In: *Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on*. IEEE. 2015, pp. 131–140.

[84] Jeffrey Svajlenko and Chanchal K Roy. "Evaluating modern clone detection tools". In: *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*. IEEE. 2014, pp. 321–330.

[85] Chanchal K Roy and James R Cordy. "NICAD: Accurate detection of near-miss intentional clones using flexible pretty-printing and code normalization".

In: *Program Comprehension, 2008. ICPC 2008. The 16th IEEE International Conference on*. IEEE. 2008, pp. 172–181.

[86] *BigCloneEval*. `https://github.com/jeffsvajlenko/BigCloneEval`. June 2017.

[87] *BigCloneBench*. `https://github.com/clonebench/BigCloneBench`. June 2017.

[88] C. S. Corley, F. Lois, and S. Quezada. "Web usage patterns of developers". In: *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 2015, pp. 381–390. DOI: 10.1109/ICSM.2015.7332489.

[89] Joel Brandt et al. "Two Studies of Opportunistic Programming: Interleaving Web Foraging, Learning, and Writing Code". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '09. Boston, MA, USA: ACM, 2009, pp. 1589–1598. ISBN: 978-1-60558-246-7. DOI: 10.1145/1518701.1518944. URL: http://doi.acm.org/10.1145/1518701.1518944.

[90] Di Yang et al. "Stack Overflow in github: any snippets there?" In: *Mining Software Repositories (MSR), 2017 IEEE/ACM 14th International Conference on*. IEEE. 2017, pp. 280–290.

[91] Rabe Abdalkareem, Emad Shihab, and Juergen Rilling. "On code reuse from stackoverflow: An exploratory study on android apps". In: *Information and Software Technology* 88 (2017), pp. 148–158.

[92] Emad Shihab et al. "An industrial study on the risk of software changes". In: *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*. ACM. 2012, p. 62.

[93] Y. Acar et al. "You Get Where You're Looking for: The Impact of Information Sources on Code Security". In: *2016 IEEE Symposium on Security and Privacy (SP)*. 2016, pp. 289–305. DOI: 10.1109/SP.2016.25.

[94] Sebastian Baltes, Richard Kiefer, and Stephan Diehl. "Attribution required: Stack Overflow code snippets in GitHub projects". In: *Proceedings of the 39th International Conference on Software Engineering Companion*. IEEE Press. 2017, pp. 161–163.

[95] Luca Ponzanelli et al. "Too Long; Didn'T Watch!: Extracting Relevant Fragments from Software Development Video Tutorials". In: *Proceedings of the 38th International Conference on Software Engineering*. ICSE '16. Austin, Texas: ACM, 2016, pp. 261–272. ISBN: 978-1-4503-3900-1. DOI: 10.1145/2884781.2884824. URL: http://doi.acm.org/10.1145/2884781.2884824.

[96] Rebecca Tiarks and Walid Maalej. "How Does a Typical Tutorial for Mobile Development Look Like?" In: *Proceedings of the 11th Working Conference on Mining Software Repositories*. MSR 2014. Hyderabad, India: ACM, 2014, pp. 272–281. ISBN: 978-1-4503-2863-0. DOI: 10.1145/2597073.2597106. URL: http://doi.acm.org/10.1145/2597073.2597106.

[97] Sebastian Baltes, Christoph Treude, and Stephan Diehl. "SOTorrent: Studying the Origin, Evolution, and Usage of Stack Overflow Code Snippets". In: *Proceedings of the 16th International Conference on Mining Software Repositories (MSR 2019)*. 2019.

[98] *BigQuery*. https://cloud.google.com/bigquery/. 2018.

[99] Manziba Akanda Nishi and Kostadin Damevski. "Scalable code clone detection and search based on adaptive prefix filtering". In: *Journal of Systems and Software* 137 (2018), pp. 130–142.

[100] H. Sajnani et al. “SourcererCC: Scaling Code Clone Detection to Big-Code”. In: *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. 2016, pp. 1157–1168. DOI: 10.1145/2884781.2884877.

[101] Sebastian Baltes et al. “The Evolution of Stack Overflow Posts: Reconstruction and Analysis”. In: *arXiv preprint arXiv:1811.00804* (2018).

[102] Joel Brandt et al. “Two Studies of Opportunistic Programming: Interleaving Web Foraging, Learning, and Writing Code”. In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI ’09. Boston, MA, USA: ACM, 2009, pp. 1589–1598. ISBN: 978-1-60558-246-7. DOI: 10.1145/1518701.1518944. URL: http://doi.acm.org/10.1145/1518701.1518944.

[103] Lingfeng Bao et al. “Tracking and Analyzing Cross-Cutting Activities in Developers’ Daily Work (N)”. In: *Proceedings of the 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. ASE ’15. Washington, DC, USA: IEEE Computer Society, 2015, pp. 277–282. ISBN: 978-1-5090-0025-8. DOI: 10.1109/ASE.2015.43. URL: http://dx.doi.org/10.1109/ASE.2015.43.

[104] Xin Xia et al. “What Do Developers Search for on the Web?” In: *Empirical Softw. Engg.* 22.6 (Dec. 2017), pp. 3149–3185. ISSN: 1382-3256. DOI: 10.1007/s10664-017-9514-4. URL: https://doi.org/10.1007/s10664-017-9514-4.

[105] Luca Ponzanelli et al. “Too Long; Didn’T Watch!: Extracting Relevant Fragments from Software Development Video Tutorials”. In: *Proceedings of the 38th International Conference on Software Engineering*. ICSE ’16. Austin, Texas: ACM, 2016, pp. 261–272. ISBN: 978-1-4503-3900-1. DOI: 10.1145/2884781.2884824. URL: http://doi.acm.org/10.1145/2884781.2884824.

[106] Tyler McDonnell, Baishakhi Ray, and Miryung Kim. "An empirical study of api stability and adoption in the android ecosystem". In: *Software Maintenance (ICSM), 2013 29th IEEE International Conference on*. ICSM '13. Eindhoven, Netherlands: IEEE, 2013, pp. 70–79. ISBN: 978-0-7695-4981-1. DOI: 10.1109/ICSM.2013.18. URL: https://ieeexplore.ieee.org/document/6676878.

[107] Deokyoon Ko et al. "API document quality for resolving deprecated APIs". In: *Software Engineering Conference (APSEC), 2014 21st Asia-Pacific*. APSEC '14. Buenos Aires, Argentina: IEEE, 2014, pp. 27–30. ISBN: 978-1-4799-7426-9. DOI: 10.1109/APSEC.2014.87. URL: https://ieeexplore.ieee.org/document/7091210.

[108] Andrew J. Ko, Robert DeLine, and Gina Venolia. "Information Needs in Collocated Software Development Teams". In: *Proceedings of the 29th International Conference on Software Engineering*. ICSE '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 344–353. ISBN: 0-7695-2828-7. DOI: 10.1109/ICSE.2007.45. URL: https://doi.org/10.1109/ICSE.2007.45.

[109] Janice Singer et al. "An Examination of Software Engineering Work Practices". In: *CASCON First Decade High Impact Papers*. CASCON '10. Toronto, Ontario, Canada: IBM Corp., 2010, pp. 174–188. DOI: 10.1145/1925805.1925815. URL: http://dx.doi.org/10.1145/1925805.1925815.

[110] André Hora et al. "How do developers react to API evolution? The Pharo ecosystem case". In: *Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on*. ICSME '15. Bremen, Germany: IEEE, 2015, pp. 251–260. ISBN: 978-1-4673-7532-0. DOI: 10.1109/ICSM.2015.7332471. URL: https://ieeexplore.ieee.org/document/7332471.

[111] Tiago Espinha, Andy Zaidman, and Hans-Gerhard Gross. "Web API growing pains: Stories from client developers and their code". In: *Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE), 2014 Software Evolution Week-IEEE Conference on*. CSMR-WCRE '14. Antwerp, Belgium: IEEE, 2014, pp. 84–93. ISBN: 978-1-4799-3752-3. DOI: 10.1109/CSMR-WCRE.2014.6747228. URL: https://ieeexplore.ieee.org/document/6747228.

[112] Anand Ashok Sawant, Romain Robbes, and Alberto Bacchelli. "On the reaction to deprecation of 25,357 clients of 4+ 1 popular Java APIs". In: *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. ICSME '16. Washington, DC, USA: IEEE, 2016, pp. 400–410. ISBN: 978-1-5090-3806-0. DOI: 10.1109/ICSME.2016.64. URL: https://ieeexplore.ieee.org/document/7816485.

[113] D. Hou and X. Yao. "Exploring the Intent behind API Evolution: A Case Study". In: *2011 18th Working Conference on Reverse Engineering(WCRE)*. Vol. 00. 2011, pp. 131–140. DOI: 10.1109/WCRE.2011.24. URL: doi.ieeecomputersociety.org/10.1109/WCRE.2011.24.

[114] Pradeep K Venkatesh et al. "What Do Client Developers Concern When Using Web APIs? An Empirical Study on Developer Forums and Stack Overflow". In: *2016 IEEE International Conference on Web Services (ICWS)*. ICWS' 16. Washington, DC, USA: IEEE Computer Society, 2016, pp. 131–138. ISBN: 978-1-5090-2675-3. DOI: 10.1109/ICWS.2016.25. URL: https://ieeexplore.ieee.org/document/7557994.

[115] Mario Linares-Vásquez et al. "How Do API Changes Trigger Stack Overflow Discussions? A Study on the Android SDK". In: *Proceedings of the 22Nd*

International Conference on Program Comprehension. ICPC 2014. Hyderabad, India: ACM, 2014, pp. 83–94. ISBN: 978-1-4503-2879-1. DOI: 10.1145/2597008.2597155. URL: http://doi.acm.org/10.1145/2597008.2597155.

[116] *Android StartActivityForResult Example*. https://www.javatpoint.com/android-startactivityforresult-example. 2018.

[117] *Android Official Documentation*. https://developer.android.com/reference/. 2018.

[118] *Android API Differences Report*. https://developer.android.com/sdk/api_diff/../changes. 2018.

[119] *Android API Differences Report For Support Library*. https://developer.android.com/sdk/support_api_diff/.../changes. 2018.

[120] *Android Version History*. https://en.wikipedia.org/wiki/Android_version_history. 2018.

[121] *NLTK 3.3 Documentation*. https://www.nltk.org/. 2018.

[122] *TextBlob: Simplified Text Processing*. https://textblob.readthedocs.io/en/dev/. 2018.

[123] *SrcML Tool Documentation*. https://www.srcml.org/. 2018.

[124] *Sklearn Metrics Precision Recall Fscore Support*. http://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_recall_fscore_support.html. 2018.

[125] *Accuracy, Precision, Recall or F1?* https://towardsdatascience.com/accuracy-precision-recall-or-f1-331fb37c5cb9. 2018.

[126] *Sklearn Metrics Precision Recall F1score*. http://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html/. 2018.

[127] *Taxicab Geometry*. https://en.wikipedia.org/wiki/Taxicab_geometry. 2018.

[128] *Random Forest Classifier*. http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html. 2018.

[129] *Multi Instance Classifier*. https://github.com/garydoranjr/misvm. 2018.

[130] Stuart Andrews, Thomas Hofmann, and Ioannis Tsochantaridis. "Multiple Instance Learning with Generalized Support Vector Machines". In: *Eighteenth National Conference on Artificial Intelligence*. Edmonton, Alberta, Canada: American Association for Artificial Intelligence, 2002, pp. 943–944. ISBN: 0-262-51129-0. URL: http://dl.acm.org/citation.cfm?id=777092.777234.

[131] URL: https://github.com/vefstathiou/SO_word2vec.

[132] Chunyin Nong et al. "FVT: a fragmented video tutor for dubbing software development tutorials". In: *Proceedings of the 41st International Conference on Software Engineering: Software Engineering Education and Training*. IEEE Press. 2019, pp. 95–99.

[133] Luca Ponzanelli et al. "CodeTube: extracting relevant fragments from software development video tutorials". In: *Proceedings of the 38th International Conference on Software Engineering Companion*. ACM. 2016, pp. 645–648.

[134] URL: https://sourceforge.net/projects/google2srt/.

[135] URL: https://ffmpeg.org.

[136] URL: https://github.com/tesseract-ocr/.

[137] Jordan Ott et al. "A deep learning approach to identifying source code in images and video". In: *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*. IEEE. 2018, pp. 376–386.

VITA

Manziba Akanda Nishi is a Ph.D. student at Department of Computer Science, Virginia Commonwealth University. She is working as research assistant in Software Improvement (SWIM) Lab. Her research interest includes the area of software engineering, machine learning, natural language processing, data mining. She completed her B.Sc. and M.S. in Computer Science and Engineering from University of Dhaka, Bangladesh.

Publications.

1. **Manziba Akanda Nishi**, and Kostadin Damevski. "Scalable code clone detection and search based on adaptive prefix filtering." Journal of Systems and Software 137 (2018): 130-142.

2. Preetha Chatterjee, **Manziba Akanda Nishi**, Kostadin Damevski, Vinay Augustine, Lori Pollock, and Nicholas A. Kraft. "What information about code snippets is available in different software-related documents? an exploratory study." In Software Analysis, Evolution and Reengineering (SANER), 2017 IEEE 24th International Conference on, pp. 382-386. IEEE, 2017

3. **Manziba Akanda Nishi**, and Kostadin Damevski. "Automatically Identifying Valid API Versions of Software Development Tutorials on the Web." Accepted in Journal of Software: Evolution and Process. July, 2019

4. **Manziba Akanda Nishi**, Agnieszka Ciborowska and Kostadin Damevski. "Characterizing Duplicate Code Snippets between Stack Overflow and Tutorials." Proceedings of the 16th International Conference on Mining Software Repositories, MSR '19, 2019, pp.240–244, IEEE Press, Piscataway, NJ, USA

5. Ashis Kumar Chanda, Swapnil Saha, **Manziba Akanda Nishi**, Md Samiullah, and Chowdhury Farhan Ahmed. "An efficient approach to mine flexible periodic patterns in time series databases." Engineering Applications of Artificial Intelligence 44 (2015): 46-63.

6. **Manziba Akanda Nishi**, Chowdhury Farhan Ahmed, Md Samiullah, and Byeong-Soo Jeong. "Effective periodic pattern mining in time series databases." Expert Systems with Applications 40, no. 8 (2013): 3015-3027.

7. Md Samiullah, Chowdhury Farhan Ahmed, **Manziba Akanda Nishi**, Anna Fariha, S. M. Abdullah, and Md Rafiqul Islam. "Correlation mining in graph databases with a new measure." In Asia-Pacific Web Conference, pp. 88-95. Springer, Berlin, Heidelberg, 2013.