Highlighting Current Issues in API Usage Mining to Enhance Software Reusability

MUSA IBRAHIM M. ISHAG
Chungbuk National University
College of Electrical and Computer Engineering
Database/Bio informatics Laboratory
Chungda-ro 1, Seowon-Gu, Cheongju
SOUTH KOREA

HYUN WOO PARK
Chungbuk National University
College of Electrical and Computer Engineering
Database/Bio informatics Laboratory
Chungda-ro 1, Seowon-Gu, Cheongju
SOUTH KOREA

DINGKUN LI
Chungbuk National University
College of Electrical and Computer Engineering
Database/Bio informatics Laboratory
Chungda-ro 1, Seowon-Gu, Cheongju
SOUTH KOREA

KEUN HO RYU
Chungbuk National University
College of Electrical and Computer Engineering
Database/Bio informatics Laboratory
Chungda-ro 1, Seowon-Gu, Cheongju
SOUTH KOREA

Abstract: The sheer amount of open source codes made available in code repositories and code search engines along with the rapidly increasing releases of Application Programming Interfaces (APIs) have made code development process easier for programmers. However, learning how to use the elements of an API properly is both challenging and requires learning curve. Mining the available client and test codes can help programmers to identify the best practices in using these APIs. In this paper, we investigate the API usage mining to identify open issues for the researchers. In particular, we make a theoretical comparison of the API usage pattern mining and highlight unresolved issues along with proper suggestions to address them.

Key–Words: API usage patterns, Mining software engineering data, association rules,frequent patterns

1 Introduction

Application Programming Interfaces (APIs) are facilitating source code reusability for programmers. Recently, the number of APIs made available to the programmers has drastically increased in different domains generating a huge number of reusable code elements. Motivated by this sheer amount of data, researchers have devised methods for mining software engineering data [1] .

A major current focus of applying data mining to software engineering data, is mining API usage patterns [9]. Researchers basically apply data mining to extract patterns that can serve both in code reusability [10] and to detect violations [14], [13].

A pattern can be considered as a violating pattern in regard to multiple factors such as a sequence of code elements that if followed can cause huge energy consumption in the device that implements it. In contrast a reusable pattern is the best practice that is usually demanded by programmers.

In order to find reusable patterns the most used and applied data mining techniques are association rule mining and clustering. A recent survey was reported in [8], where the authors have empirically evaluated the efficiency of applying itemset mining and sequential pattern mining to the problem of mining call-usage patterns. More recently, Shaheen and Azhar [9] have reviewed source code mining techniques where they have categorized the techniques into three general categories; namely, programming rules, copy-paste detection, and API usage.

In this paper, the most used techniques of API usage pattern mining are investigated in order to help researchers progress in this direction. In essence, the paper describes the general framework of mining API usage patterns, evaluates the techniques used, and highlights the current issues along with viable suggestions for addressing them. Therefore the key contributions of this paper can be summarized in the following:

- Theoretical comparison and evaluation of the API usage pattern mining techniques.

- Highlight issues in the existing systems and suggest possible enhancements.

The following section defines the problem of API usage mining categorizing it into three classes. Namely, frequent itemset based, frequent sequence based, and graph based techniques. Afterwards, a theoretical comparison is given along with a highlight about the current issues and possible suggestions. Finally, a summary concludes this article.

2 API Usage Pattern Mining

A good practice in software development is to produce reusable source codes. This goal is achieved at its best by means of APIs. However, the most challenging task facing software developers is the learning curve required in order to accomplish their coding tasks using APIs. Many reasons combined attribute to this challenge including lack of proper documentation, shortages in publicly available client codes-codes that use elements of the API, in addition to the rapid increase in the newly published APIs.

The smart move by researchers to utilize data mining as a mean for discovering these reusable patterns [1],[15] has resulted in a number of available tools for software engineers. In order to better understand these tools a definition to the problem is given bellow.

2.1 Problem Definition

Generally, the problem of API usage pattern mining can be defined as the process of finding the proper usage sequence or order of a group of reusable code elements within an API. In this sense, the frequent pattern mining [16]-a step in association rule analysis [29], and data clustering are inevitable.

To illustrate the process of frequent usages of API, consider the hypothetical code example in Figure 2. The local methods of the client code are the transactions. Whereas, the code elements represented as methods calls can be considered as items in a traditional market basket analysis. The goal will be to find method calls that frequently occur together in order to form implication rules. From the example the following taxonomies are demanded:

Support count of a code element is the number of times it occurs in code elements of a client or test codes. (i.e transactions).

API-usage pattern (call-usage pattern) is an implication relation.

Support of an API-usage rule is the support of its elements.

Confidence of an API-usage rule is the relative occurrence of the code elements contained in the rule. For the rule to be significant, it must both be frequent by satisfying a user provided *minimum support* and confident by satisfying a *minimum confidence threshold*.

Figure 1 describes the general framework for API usage mining where the data sets are constituted from collections of client codes available on the web which can be gathered by querying code search engines, and test codes that might be found on API documentation files. An example of popular search engines is provided in table 1.

Based on the way these data sets are preprocessed, three paths of mining techniques can be distinguished. Namely; frequent itemset based, frequent sequence based and frequent graph based methods.

2.2 Frequent Itemset based Methods

An algorithm that follows this approach formulates the problem by applying a typical analogy of the traditional frequent itemset mining to mine the frequent usage patterns. In essence the source code data set must be preprocessed and converted into transactions containing items. That is local methods represent transactions and API methods called within these methods represent items. Afterwards, traditional itemset mining algorithms (Apriori [29], FP-growth [30]) can be applied to discover the patterns and formulate rules. Figure 2.b how a source code data set converted into a traditional market basket transaction data.

2.3 Frequent Sequence based Methods

In this case, algorithms following this approach preprocess the source code data set and convert it into sequences of API method calls where the co-occurrences and the order of calls matter. Thereafter, the task becomes finding frequently occurring ordered sequences of method calls. Therefore, traditional frequent sequence mining algorithms can possibly be applied. This process is shown in figure 2.c.

2.4 Frequent Graph based Methods

Methods following this approach model the API method call sequences as directed acyclic graphs. Therefore, in the preprocessing step the source code data will have to be converted into call sequence graphs or subgraphs. The task will then be looking for frequent graphs in order to formulate rules. Figure 2.d illustrates this conversion.

The discovered patterns and association rules are usually incorporated into Integrated Development Environments (IDEs) in order to help in code suggestions. Although this is not fully realized currently, it will lead to a new generation of intelligent IDEs which

Name	Ref	Link
Search code	[17]	https://searchcode.com/
Black Duck Open Hub	[18]	http://code.openhub.net/
Codase	[19]	http://www.codase.com/
Google code	[20]	https://code.google.com/
Krugle	[21]	http://www.krugle.com/
F1Sourcecode	[22]	http://www.f1sourcecode.com/
Nerdy Data	[23]	http://nerdydata.com/
Symbol hund	[24]	http://www.symbolhound.com/
Meanpath	[25]	https://meanpath.com/

Table 1: Popular Code Search Engines.

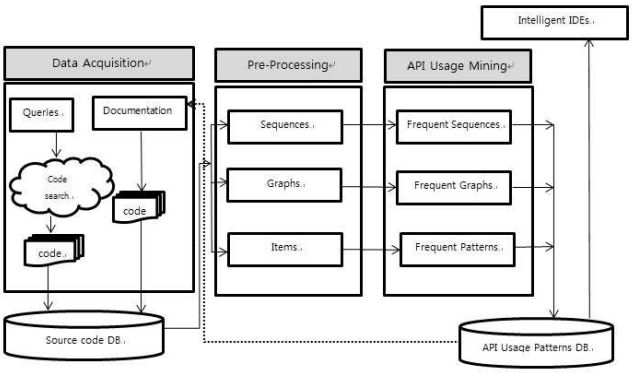

Figure 1: A general Framework for Mining API-Usage Patterns.

will be capable of providing extra functionalities to the software developers using them as it is illustrated in figure 1.

3 Comparing The Methods

In this paper, we have considered the most popularly cited works and tools available in the literature. These include CodeWeb[2], Strathcona [3], Prospector [4], XSnippet [5], MAPO [6], and ParseWeb[7] which have previously been studied and compared from different prospective by Shaheen and Azhar [9].

We have additionally considered most recent studies like UsETeC [10], and eXoaDoc [11],[12]. The algorithms are compared according the following three dimensions:

- **Data Sources**
  Based on the data set used in the mining process, the current tools fall into two basic categories. Those using client codes from the internet through issuing queries to code search engines as explained in table 1, and others that utilize the test examples from the associated documentations. From the comparison shown in table 2, CodeWeb, Strathcona, Prospector, XSnippet, MAPO, and ParseWeb use client codes from the web. Whereas, only UsETeC and eXoaDoc exploit the test examples. In essence, eXoaDoc helps in adding proper test code examples to the API documentation.

- **Patterns**
  The patterns extracted can also fall in the three general categories explained in figure 2. Among the studies considered in this paper, MAPO, ParseWeb, and UsETeC search for sequential patterns. Prospector and XSnippet find graph patterns. Whereas, only CodeWeb is searching for frequent items.

- **Algorithmic Approach**
  Some of the algorithms consider the data sets are stored in external storage devices. Therefore, a scan is performed to read the data from the disc to memory. Whereas, others consider data structures like trees to compress and store the entire data set in memory and perform the mining. All algorithms except MAPO need to read the data from external discs.

4 Open Issues

Based on the above comparisons we can distinguish the following as issues and directions for researchers to investigate. In essence we categorize them into four main classes. Namely, the data sets, scalability, algorithms and tools.

- **Data sources** although some data sets are available for researchers which include source code repositories and search engines, still the problem of getting representative data needs to be studied. In particular, new direction is emerging that tries to enrich the documentations of the newly release APIs with test examples. UsETeC [10] and eXoaDoc [11, 12] are leading this direction.

- **Scalability** the algorithms and tools developed so far need to be re-engineered in order to scale to the increasing release of new software and APIs. A possible suggestion here is to exploit the capabilities of BIG DATA[26] tools and technologies. Therefore, new scalable algorithms can be based on Hadoop[27] and MapReduce[28] programming model.

- **Algorithms** The way the current algorithms are developed is solely based on the assumption of key-value representation of the code elements. That is the algorithms consider the existence or absence of an item. However, in real world example of software development, occurrences of

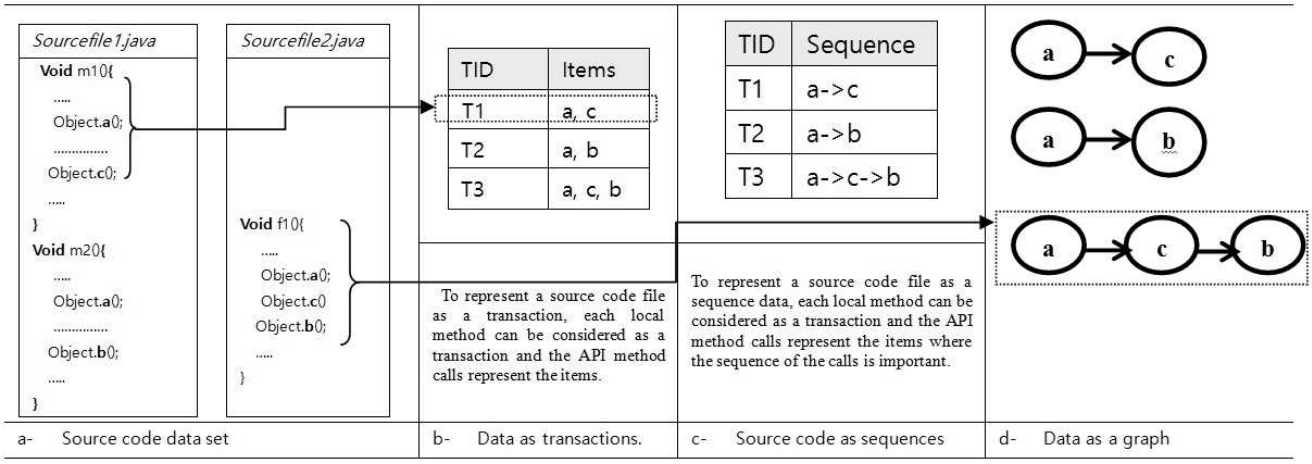

Figure 2: Hypothetical Source Code Dataset and its possible representations

Method	Ref.	Data Source		Patterns			Algorithmic Approach		Issues
		Client	Test	Sequences	Graphs	Pattern	Scan DB	Compress	
CodeWeb	[2]	√	-	-	-	√	√	-	-domain specific
Strathcona	[3]	√	-	-	-	-	√	-	Eclipse dependent
Prospector	[4]	√	-	-	√	-	√	-	-accuracy
XSnippet	[5]				√		√		-dependent on queries
MAPO	[6]	√	-	√	-	-	-	√	-mines closed patterns. -yet to be integrated in IDEs
ParseWeb	[7]	√	-	√	-	-	-	-	-needs to be integrated into IDEs
UsETeC	[10]	-	√	√		-	√	-	-Library dependent
eXoaDoc	[11],[12]	-	√	-	-	-	√	-	

Table 2: Theoretical Comparison of Popular Methods.

code elements might have different significant.

For example, one can distinguish system calls from regular method calls. Therefore, sophisticated algorithmic approaches that reflect these asymmetric relationships are needed. A possible solution is to adapt more advanced pattern mining approaches. Where weighted frequent patterns and utility based mining might inspire researchers to tackle this issue.

- **Need for tools and intelligent IDEs** As explained in figure 2, a possible utilization of the discovered patterns is to integrate them into IDEs. This might lead to new generations of intelligent IDEs that might suggest not only a code completion but also would suggest examples.

Another direction is the lack of dedicated web and cloud services that might help providing best practices as a service for clients (i.e software developers). A possible use case of such services would be the ability for software developers to

submit their source codes for investigation before the actual release. Thus, it can be considered a great contribution towards intelligent software testing.

Addressing the above mentioned issues will result in an advanced practice in software engineering both in the reusability, and security and testing.

5 Conclusion

The development of reusable software is one of the corner stones of software development. Towards this direction, a plethora of open source codes are available and being circulated online. APIs are the core of this reusable software. However, to reduce the learning curve spent by a programmer in learning how to use the code elements of these APIs, researchers have

adopted data mining as a solution. In this paper, a theoretical evaluation and comparison was conducted comparing the most popular and current tools available. The comparison considered three main dimensions. Namely the source of the data set used the type of API-usage patterns discovered, and the algorithmic approach followed by these tools.

Concurrently, the paper has distinguished three issues to further the API-usage mining research. In particular, the issues of data sources, scalability, and algorithms and the need for tools and intelligent IDEs are still open for research and contributions. To this end, the paper has given possible suggestions. c.

Acknowledgements: This research was supported by the MSIP(Ministry of Science, ICT and Future Planning), Korea, under the ITRC(Information Technology Research Center) support program (IITP-2015-H8501-15-1013) supervised by the IITP(Institute for Information & communication Technology Promotion), and by Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT & Future Planning (No.2013R1A2A2A01068923).

References:

[1] A. Hassan, and T. Xie, *Mining software engineering data,* in Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 2, 2010, pp. 503-504.

[2] A. Michail, *Data mining library reuse patterns using generalized association rules,* in Proceedings of 22nd International Conference on Software Engineering (ICSE'00), Limerick, Ireland, 2000, pp 167-176.

[3] R. Holmes, and G. C. Murphy, *Using structural context to recommend source code examples,* in Proceedings of the 27th international conference on Software engineering, 2005, pp. 117-125. .

[4] D. Mandelin, L. Xu, R. Bodk et al., *Jungloid mining: helping to navigate the API jungle,* ACM SIGPLAN Notices, vol. 40, no. 6, pp. 48-61, 2005.

[5] N. Sahavechaphan, and K. Claypool, *XSnippet: mining for sample code,* ACM SIGPLAN Notices.

[6] T. Xie, and J. Pei, *MAPO: Mining API usages from open source repositories,* in Proceedings of the 2006.

[7] S. Thummalapenta, and T. Xie, *Parseweb: a programmer assistant for reusing open source code on the web,* in Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering, 2007, pp. 204-213. international workshop on Mining software repositories, 2006, pp. 54-57. vol. 41, no. 10, pp. 413-430, 200.

[8] Kagdi, Huzefa, Michael L. Collard, and Jonathan I. Maletic. *Comparing approaches to mining source code for call-usage patterns,* Mining Software Repositories, 2007. ICSE Workshops MSR'07. Fourth International Workshop on. IEEE, 2007.

[9] Khatoon, Shaheen, Azhar Mahmood, and Guohui Li. *An evaluation of source code mining techniques,* Fuzzy Systems and Knowledge Discovery (FSKD), 2011 Eighth International Conference on. Vol. 3. IEEE, 2011.

[10] Zhu, Zixiao, et al. *Mining api usage examples from test code,* Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on. IEEE, 2014.

[11] Kim, J., Lee, S., Hwang, S. W., and Kim, S. *Adding examples into java documents,* In Proc. of ASE09. pp. 540-544.

[12] Kim, J., Lee, S., Hwang, S. W., and Kim, S. *Enriching Documents with Examples: A Corpus Mining Approach,* ACM Transactions on Information Systems (TOIS), 31(1) (2013), pp.

[13] Linares-Vsquez, Mario, et al. *Mining energy-greedy API usage patterns in Android apps: an empirical study,* Proceedings of the 11th Working Conference on Mining Software Repositories. ACM, 2014.

[14] Aafer, Yousra, Wenliang Du, and Heng Yin, *DroidAPIMiner: Mining API-level features for robust malware detection in android,* Security and Privacy in Communication Networks. Springer International Publishing, 2013. 86-103.

[15] Mendez, Diego, Benoit Baudry, and Martin Monperrus, *Analysis and Exploitation of Natural Software Diversity: The Case of API Usages,* Diss. Inria, 2014.

[16] Han, Jiawei, Micheline Kamber, and Jian Pei. *Data mining,* southeast asia edition: Concepts and techniques. Morgan kaufmann, 2006.

[17] Search Code, https://searchcode.com/

[18] Black Duck Open Hub, http://code.openhub.net/

[19] Codase Site, http://www.codase.com/

[20] Google Code, https://code.google.com

[21] Krugle, http://www.krugle.com/

[22] F1 Source Code, http://www.f1sourcecode.com/

[23] Nerdy Data, http://nerdydata.com/

[24] Symbol Hund, http://www.symbolhound.com/

[25] Mean Path, https://meanpath.com/

[26] Manyika, James, et al., *Big data: The next frontier for innovation,* competition, and productivity, 2011.

[27] White, Tom. *Hadoop: the definitive guide: the definitive guide.* O'Reilly Media, Inc., 2009.

[28] Dean, Jeffrey, and Sanjay Ghemawat, *MapReduce: simplified data processing on large clusters,* Communications of the ACM 51.1 (2008): 107-113.

[29] Agrawal Rakesh, and Ramakrishman Srikant. *Fast Algorithms for Mining Association Rules in Large Databases,* In VLDB, 1994.

[30] Jiawei Han, Jian Pei, and Yiwen Yin. *Mining frequent patterns without candidate generation,* In SIGMOD, 2000.