# On the Variations and Evolutions of API Usage Patterns: Case Study on Android Applications

Koki Ogasawara
k-ogaswr@ist.osaka-u.ac.jp
Osaka University
Suita, Osaka, Japan

Tetsuya Kanda
t-kanda@ist.osaka-u.ac.jp
Osaka University
Suita, Osaka, Japan

Katsuro Inoue
inoue@ist.osaka-u.ac.jp
Osaka University
Suita, Osaka, Japan

## ABSTRACT

Software developers can reduce the implementation cost by calling already provided functions through accessing library Application Programming Interface (API). APIs are often used in combination but how to combine them are not well-documented. Existing researches focused on how to extract API usage patterns or how to detect API misuse from existing software. This kind of research might be affected by dataset to analyze, so to improve mining results and to understand how the difference of API usage patterns affect the software health are important tasks. We conducted an analysis on variations of API usage pattern among software projects and their version history with Android SDK APIs and Android applications. Based on our analysis results, we made some suggestions for further API analysis. For example, there are many project-specific API usage patterns and long-life uncommon API usage patterns so that they might affect the mining result or checking software health status.

## CCS CONCEPTS

• **Software and its engineering** → **Maintaining software**; **Software evolution**.

## KEYWORDS

APIs, API patterns, Android

**ACM Reference Format:**
Koki Ogasawara, Tetsuya Kanda, and Katsuro Inoue. 2018. On the Variations and Evolutions of API Usage Patterns: Case Study on Android Applications. In *acmBooktitle.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/1122445.1122456

## 1 INTRODUCTION

The Application Programming Interface (API) is a set of functions to communicate with libraries or platforms from client applications. Software developers can reduce the implementation cost by calling already provided functions through accessing library APIs [9, 14]. Well-maintained and widely-used APIs are reliable because maintainers of APIs and many users can find bugs or implement new features. Users of APIs can receive these benefits by updating libraries. Thus, using reliable APIs also contributes to keeping client applications' good health.

APIs are often used in combination to implement a feature. So the key to help developers to write a complex code and to avoid reinventing the wheel is identifying how to combine APIs in existing software. To support developers to learn how to use APIs in combination, repository mining technique is applied to extract API usage patterns [27].

API misuse is a major cause of failure of software, including vulnerability and performance bugs [5, 13]. However, APIs are often used as variations of major patterns. For example, some APIs have an optional method which developer can choose whether they use or not. In some cases instance of the object is passed as an argument so that a method to generate an instance is not called. Considering these situations, minor API usage patterns are not always the misuse. Another concern is that the reliability of API usage patterns extracted automatically are depends on the analysis target [28]. If a project in the dataset contains a large number of specific API usage patterns, it becomes the major pattern and will affect the extract result regardless of whether it is a project-specific pattern or not. Despite these problems, it is not revealed that whether the difference of software development projects affect the API usage patterns.

We should also focus on the API uses in the history in the software development project. The majority of API usage patterns can be changed over time according with updating software. In addition, libraries are also upgraded and deprecated or newly introduced APIs will override existing API usage patterns. Thus, mining-based approaches will be affected by target versions they choose. In addition, if the minor API usage patterns survive for a long time, it may not have affected a significant impact to the software health.

To improve mining results and to understand how the difference of API usage patterns affect the software health, we conducted an analysis on variations of API usage pattern among software projects and their version history. In this paper, we conducted a case study with Android SDK as a source of APIs and Android applications as an analysis target, and conducted project-wide and historical analysis to reveal the difference of API usage patterns among software development projects and throughout their evolution history.

The followings are the summary of our case study results.

- While some of API usage patterns are frequently used but there are many uncommon API usage patterns.
- Most of the API usage patterns consist of frequent API usage pattern with additional API calls. However, there are still many API usage patterns that are not the variant of frequent API usage patterns.


*SoHeal 2020, acmConference, acmConference*

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

- Frequent API usage patterns among projects are the major part of patterns, but there is an API that most of API usage patterns are project-specific. It depends on APIs.
- Frequent API usages are widely used in various projects, even though most of API usage patterns are project-specific.
- With the new releases of the project, the number of API usage patterns increases. However, not only an increase in API usage patterns but also a decrease in API usage patterns occur at the same time in most of the projects.

We describe related woks about API usage pattern mining and analysis of APIs in Section 2. The case study is described in the Section 3 and it includes more detailed results. And finally, we conclude and make a suggestion for further API research in Section 4.

## 2 RELATED WORKS

API documentation is an important resource to find out how to combine APIs to archive implementing a feature. However, documentation often lacks information and it is difficult to learn API usages for developers of client applications. For example, learning resources such as documentation do not describe enough information, or each API call is easy to understand but it is not clear that how to combine APIs to archive the developers' task [8, 16, 17, 25].

To support developers to learn how to use APIs, various approaches are proposed based on mining existing codebase. MAPO [27] is a method that extracts API usage patterns from source code. Given the method name or the class name of API as a query, MAPO collects source files related to the query and extract frequently appearing API uses from them. Their result contains redundant usages, so UP-Miner [24] introduced two-steps clustering to improve mining results. They also proposed metrics to evaluate the API usage mining result. MLUP [19] also detects the pattern of API calls. This method focuses on the co-relationship with API methods to clustering so that users can distinguish APIs always used together or used in specific situations. AST-based API usage mining approach is also proposed by Lämmel et al.

Online resources such as Q&A forum and blog posts are also important learning and mining resources [18, 23]. Instead of mining API usages from complete source code, Azad et al. proposed a predictive model to predict changes in API usages from software development history and Stack Overflow posts [2]. Library itself is also combined with another library to use. LibRec [21] and LibCUP [20] focuses on usage patterns of libraries to indicate which set of libraries are co-used together to developers.

However, collecting API usage examples is a hard task, especially for mining good examples or newly released libraries. Fowkes et al. proposed a tool named PAM [6], a near parameter-free probabilistic algorithm. PAM employs a machine learning technique to find out the most interesting API call patterns. They also reported that handwritten examples do not cover real API usages, only 27% coverage in their experiment.

Lack of API documentation leads developers to misuse API. API misuse is a major cause of failure of software, including vulnerability and performance bugs [5, 13]. Zhang et al. studied code snippets on Stack Overflow and found out that almost one-third of posts may contain potential API usage violations [28]. To deal with the API misuse problem, some researches focus on automatically finding misuse of method calls [1, 10, 15]. The mutation analysis technique is introduced to this area and its evaluation result achieves high precision in finding API misuses [26].

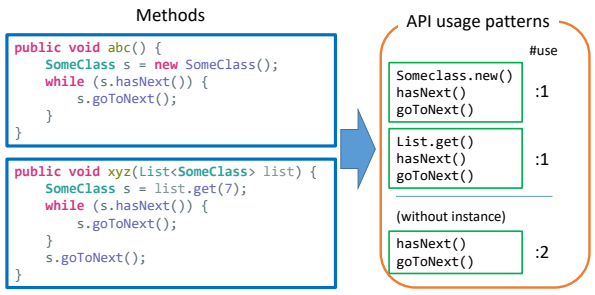


**Figure 1: An example: How to extract API usage patterns**

In this research, we investigated APIs from the Android Software Development Kit (Android SDK) [4]. Android [7] is one of the most popular platforms for mobile phones and tablets. Android application developers use the Android SDK which includes APIs to control a camera, a GPS device, a touch screen and so on. The Android SDK and Android applications are very popular and widely used for many areas of software engineering research [3, 11, 22]. The point of view of focusing on Android APIs and its usage, the work by McDonnell et al. is a very important research [12]. They analyzed API evolutions of the Android SDK and how client applications follow the updates. One big difference between their research and our case study is that we are focusing on the changes in client application by investigating a set of APIs while they were mainly focusing on changes in the SDK.

## 3 CASE STUDY

We conducted a project-wide and historical analysis to reveal the difference of API usage patterns among software development projects and throughout their evolution history.

In this paper, we use the terms "API usage pattern" and "#use of the pattern". The term **API usage pattern** is the sequence of API calls. It is extracted from each method of application code and contains the sequence of API calls of a specific class in the Android SDK. The number of API usage patterns (**#pattern**) represents how many API usage patterns are there in the project. The term #use of the pattern (**#use**) represents how many instances of API usage patterns appear in the project.

We conducted the case study to answer the following research questions.

- RQ1 How many API usage patterns are contained in Android application projects and what is the difference between frequent API usage patterns and uncommon ones?
- RQ2 What is the difference in API usage patterns among projects?
- RQ3 How the new releases of applications affect API usage patterns and their #use?

### 3.1 Extracting API Usage Patterns

Our methodology to extract API usage patterns is based on the static analysis. Figure 1 shows a small example of how to extract API

**Table 1: Target APIs**

| API | Function |
|---|---|
| `android.content.Context` | A global information about an application environment |
| `android.content.Intent` | Interface to pass information among applications |
| `android.database.Cursor` | Accessor to the search result of database |

**Table 2: Dataset**

| Project name | LOC (latest version) |
|---|---|
| AmazeFileManager | 47,065 |
| AndroidUtilCode | 42,116 |
| bitcoin-wallet | 24,905 |
| cgeo | 109,288 |
| codenameone | 591,149 |
| collect | 75,790 |
| ExoPlayer | 195,539 |
| fresco | 109,099 |
| k9 | 125,541 |
| mapbox | 40,437 |
| news-android | 20,828 |
| owncloud | 63,162 |
| owntracks | 14,812 |
| photoview | 2,305 |
| realm-java | 136,333 |
| RedReader | 45,837 |
| robolectric | 4,972 |
| Signal-Android | 111,234 |
| skytube | 20,083 |
| wikipedia | 76,402 |
| WordPress | 151,074 |
| Total | 2,052,505 |

usage patterns from two methods. We got two methods `abc()` and `xyz()` using `SomeClass`. Firstly, we find the instance of `SomeClass`. In both methods, we can see that a variable `s` is created. Then we extract the method calls of `s` in order of appearance in the text. Some API calls are included in the loop or are used repeatedly. However, counting the loop makes many minor API usage patterns with a non-essential difference so we extract only the first appearance of each API call. Now we extracted two API usage patterns in the upper right of the figure. Both API usage patterns appear once, so their #use are both 1.

While conducting this research, we have noticed that many API usage patterns have their variants which have common subsequence of API calls. We found that they are mainly caused by their different initial method call to getting an instance. For example, getting an instance from collections or creating a new instance. Therefore, we also examined API usage patterns without methods that getting or generating an instance (hereinafter referred to as *without instance*). Following this method, we ignore the methods that getting or generating an instance then only one API usage pattern is extracted (lower right).

**Table 3: API usage patterns we extracted**

| API class name | #pattern | (w/o instance) | #use |
|---|---|---|---|
| android.content.Context | 44 | 18 | 218 |
| android.content.Intent | 213 | 144 | 1233 |
| android.database.Cursor | 206 | 136 | 380 |

## 3.2 Analysis Target

In the case study, we analyzed Android APIs and their usage patterns in client applications. In detail, we analyze API usage patterns of 3 Android APIs. Table 1 shows the list of target Android APIs. We have collected 21 Android application projects from GitHub to mining API usage patterns. For each project, we picked up versions that released every six months from January 2016 and collected six versions. For each version, we analyzed Java source files to extract API usage patterns. The list of projects and their LOC at the latest version are shown in Table 2.

Before the detailed analysis, we explored how many API usage patterns and their #use there are in our dataset. Table 3 shows the result. Both `Intent` and `Cursor` have a large number of API usage patterns while `Context` has a limited number of API usage patterns despite their total #use is not so small.

## 3.3 RQ1

*How many API usage patterns are contained in Android application projects and what is the difference between frequent API usage patterns and uncommon ones?*

In RQ1, we focus on the frequency of API usage patterns and reveal how the API usage pattern appears in client projects. We analyzed the latest versions of each Android application project in this part.

We investigated the distribution of #use. Figures 2, 3, and 4 are the frequency distribution graph of each API usage patterns. The horizontal axis represents #use, and the vertical axis represents #pattern. Their common points are that most of the API usage patterns are used 5 times or less but some of the API usage patterns have a large number of #use. `Context` for example, top-5 API usage patterns account for 65% and uncommon 36 API usage patterns whose #use is 5 or less account for 25% of total #use.

Table 5 shows the top-5 API usage patterns for each APIs. We can see that most of API usage patterns are only getting instance or appending single API call with getting instance. Table 6 shows the top 5 API usage patterns without instance. This result does not contain the method calls to getting instance but these API usage patterns also consist of one or two API calls.

As we can see that frequent API usage patterns contain a few API calls, we made a hypothesis that uncommon patterns are the variants of frequent patterns. The variants of API calls are classified

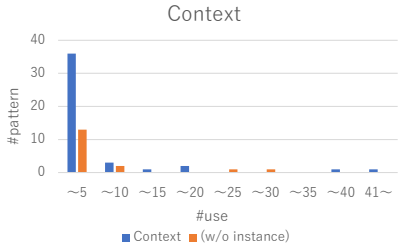


**Figure 2: Frequency distribution of `Context` API usage patterns**

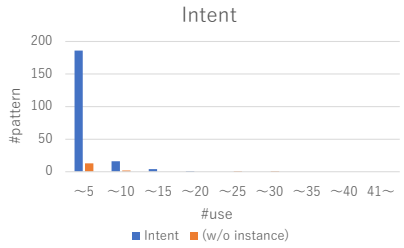


**Figure 3: Frequency distribution of `Intent` API usage patterns**

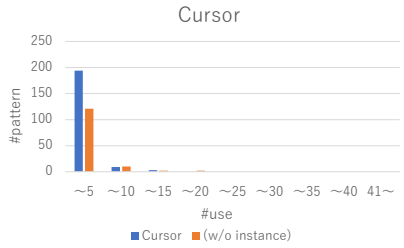


**Figure 4: Frequency distribution of `Cursor` API usage patterns**

**Table 4: Variants of frequent API usage patterns**

| API | Frequent pattern (# use >=10) | expand | sub | not a variant |
|---|---|---|---|---|
| Intent | setAction/ | 30 | - | 52 |
| | setAction/putExtra/ | 15 | 2 | |
| | setData/ | 15 | - | |
| | putExtra/ | 58 | - | |
| | addFlags/ | 20 | - | |
| | setType/putExtra | 14 | - | |
| | setClass/setAction | 1 | 1 | |
| | setType/ | 20 | - | |
| | putExtra/addFlags/ | 10 | 2 | |
| | putExtras/ | 1 | - | |
| | setDataAndType/ | 14 | - | |
| Cursor | moveToFirst/ | 67 | - | 15 |
| | moveToNext/getString/ | 40 | 2 | |
| | moveToFirst/close/ | 43 | 2 | |
| | getCount/ | 28 | - | |
| | moveToNext/ | 64 | - | |

into *expand* which represents the API usage pattern is based on the frequent API usage pattern and some more API calls are appended, and *sub* which represents the API usage pattern is a subset of the frequent API usage pattern. We firstly extracted frequent patterns (without instance) whose #use is 10 or higher, and then we investigated how many patterns are their variants.

Table 4 shows the result. A column "not a variant" means they are neither frequent API usage pattern nor their variants. `Context` has a small number of API usage patterns so it is excluded from this table. From this result, most of the frequent API usage patterns have 10 or higher their expanded variants and there are few subsets of the frequent API usage patterns. In this table, we can see that 8 of 9 subsets of the frequent API usage patterns also appear as frequent API usage patterns. The reason why `Intent` has many non-variant API calls is that frequent API usage patterns consist of sending a message like `setSomething` or `putSomething` so that APIs getting a message are considered as uncommon patterns in this case study.

Here we conclude the result of RQ1:

- While some of API usage patterns are frequently used but there are many uncommon API usage patterns. Top-5 API usage patterns account for 65% of #use while about to 36 million API usage patterns are five or fewer #use in our dataset.
- Most of the API usage patterns consist of the frequent API usage pattern with additional API calls. However, there are still many API usage patterns that are not the variant of frequent API usage patterns.

## 3.4 RQ2

*What is the difference in API usage patterns among projects?*

In RQ2, we focus on the difference among the client projects. We analyzed the latest versions of each Android application project in this part.

We firstly investigated how many projects use each API usage pattern. Figures 5, 6, and 7 show the results. Each dot represents an API usage pattern and shows how many projects using it and how many #use in total. In `Context` and `Intent`, the larger number of projects using API usage patterns, the larger #use in total and their correlation coefficient are greater than 0.9. We should note that there are also cases that an API usage pattern appears in only one project but its #use is larger than 10, so there is some dispersion. `Cursor`, on the other hand, the scatter plot is dispersed and no strong correlation with a number of projects and their #use.

**Table 5: Top 5 API usage patterns**

| Rank | Context | Intent | Cursor |
|---|---|---|---|
| 1 | getContext/ | Intent/putExtra/ | query/ |
| 2 | getApplicationContext/ | Intent/ | rawQuery/moveToFirst/ |
| 3 | getContext/getString/ | Intent/setAction/putExtra/ | query/moveToNext/ |
| 4 | getContext/getResources/ | Intent/setData/ | rawQuery/ |
| 5 | getActivity/ | Intent/setAction/ | rawQuery/moveToFirst/moveToNext/ |

**Table 6: Top 5 API usage patterns (without instance)**

| Rank | Context | Intent | Cursor |
|---|---|---|---|
| 1 | getString/ | putExtra/ | moveToFirst/ |
| 2 | getResources/ | setAction/putExtra | moveToFirst/close |
| 3 | startActivity/ | setData | moveToNext |
| 4 | getPackageName | setAction/ | moveToNext/getString |
| 5 | getSystemService/ | addFlags | getCount/ |

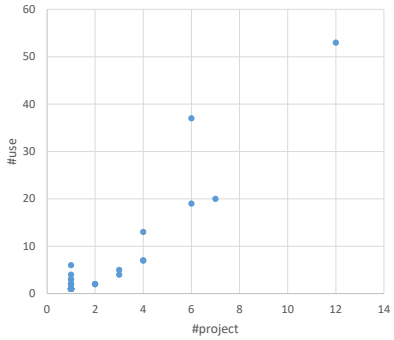


**Figure 5: How many projects use each `Context` API usage patterns**

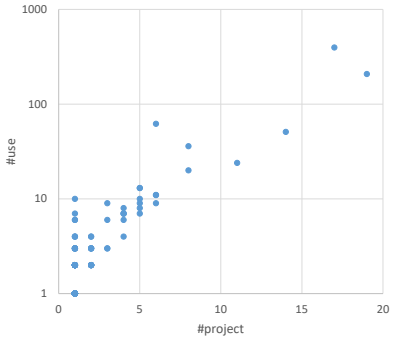


**Figure 6: How many projects use each `Intent` API usage patterns**

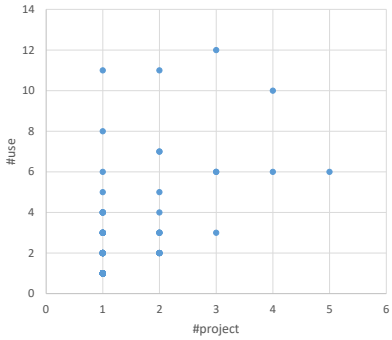


**Figure 7: How many projects use each `Cursor` API usage patterns**

Especially, some of the frequent API usage patterns appear in only one project and those project-specific API usage patterns account for a large percentage in the frequent API usage patterns.

Secondly, we investigated whether all of the projects in the dataset use project-specific API usage patterns or not. We counted #API use for top-5 API usage patterns and project-specific API usage patterns. Table 7 shows the result. In `Context` and `Intent`, top-5 API usage patterns are used over 60% in total #use and also widely used in various projects especially in the project which total #use is large. On the other hand, most of the projects have their project-specific API usage patterns and it is 20% in total. In `Cursor` we got a different result from those two APIs. Top-5 API usage patterns appear in only 13% in total and over 70% of #use are project-specific.

Here we conclude the result of RQ2:

- In some APIs, frequent API usage patterns among projects are the major part of patterns, but there is an API that most of API usage patterns are project-specific. It depends on APIs.
- Frequent API usages are widely used in various projects, even though most of API usage patterns are project-specific.

## 3.5 RQ3

*How the new releases of applications affect API usage patterns and their #use?*

In RQ3, we focused on the changes of API usage patterns and their #use in the evolution history of client applications. We analyzed all versions in the dataset.

Figure 8 represents the transition of `Context` API usage patterns and their #use with six versions we extracted by stacked line chart. The horizontal axis represents the versions, and the vertical axis represents #use. Each color represents API usage pattern and its width along with vertical axis represents #use of the API usage pattern. Frequent API usage patterns in the old version are also frequent in the new version and their #use are keep increasing. Uncommon API usage patterns, on the other hand, tend to repeat the little increase and decrease. Figure 9 shows those uncommon API usage patterns in detail. In many uncommon API usage patterns, #use shifts from zero to a few or from a few to zero. `Intent` and `Cursor` also follow a similar pattern.

Table 8 shows #pattern for each project comparing the oldest and the newest versions in the dataset. Basically, #pattern is increased from the old version to the new version, 1.5 times larger

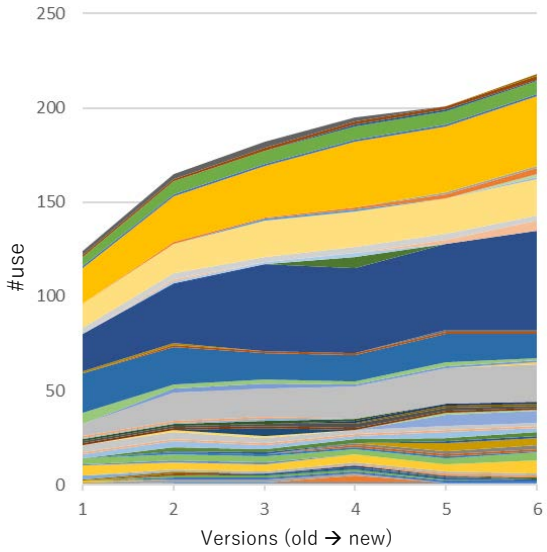


**Figure 8: `Context` API usage patterns with version history**

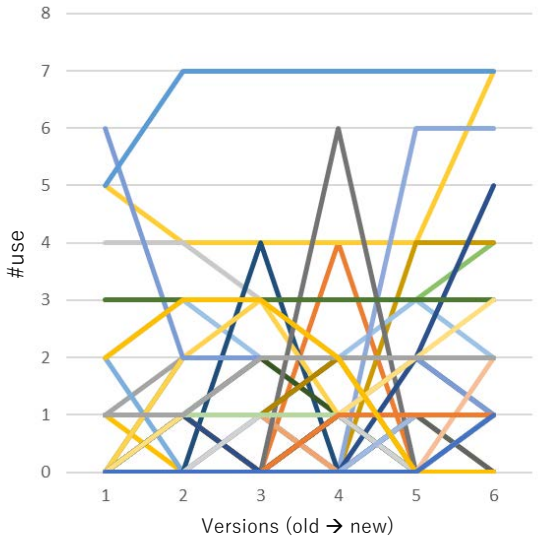


**Figure 9: Uncommon `Context` API usage patterns with version history**

in total. Source code is likely added and becomes larger in the software development projects, and #pattern also becomes larger accordingly.

However, when we count #pattern without instance, it is not increased greatly. In `Cursor` there are a little increase and in `Intent` and `Context` there are almost the same #pattern among old and new versions. Looking into the newly appeared API usage patterns, 5 of 17 in `Context`, 34 of 55 in `Intent`, and 32 of 65 in `Cursor` are caused by applying new method call to get an instance to existing API usage patterns. In addition, many method calls to get an instance are project-specific, 4 in `Context`, 53 in `Intent`, and 47 in `Cursor`. We can see that there are project-specific manners to get an instance and they affect the mining result of API usage pattern and its historical analysis.

We also investigated how #use increases and decreases for each project. As we described above, a method call to getting instance is mostly project-specific so we analyzed API usage patterns without instance. Figures 10, 11, and 12 show the increase and decrease of API usage patterns while upgrading to new versions. The horizontal axis represents the decreases, and the vertical axis represents increases. Each dot represents each new release. From those plots, not only an increase of API usage pattern but also a decrease of API usage pattern occurs at the same time, while increases are much larger than decreases. This result matches the phenomenon we analyzed in Figure 9, API usage patterns newly appeared and disappeared with new releases.

Here we conclude the result of RQ3:

- Frequent API usage patterns in the old version tend to increase their #use with new releases. Uncommon API usage patterns, on the other hand, newly appeared and disappeared with new releases.
- With the new releases of the project, #pattern increases. However, not only an increase in API usage patterns but also a decrease in API usage patterns occur at the same time in most of the projects.
- The increase of API usage pattern is mainly caused by applying a new method call to get an instance to the existing API usage pattern, but this tendency depends on the API.

## 4 CONCLUSION

In this paper, we investigated API usage patterns and its #use with 3 APIs from the Android SDK and 24 client applications. From our result, we can suggest the following topics.

- In some APIs, there are many project-specific API usage patterns so API usage patterns extracted by mining will be strongly affected by the target dataset. Popularity-based misuse detection also might be affected.
- Some uncommon API usage patterns increase or decrease at the same period among multiple applications. This kind of trend would helpful for understanding the reliability of API usage pattern.
- Most of the API usage patterns are consist of the frequent API usage pattern with additional API calls and subsets of the frequent API usage patterns also appear as frequent API usage patterns. We should consider such variants when mining API usage patterns.

We did not analyze whether such kind of variations of API usage patterns are harmful or not, so we would like to expand this project-wide and historical approach to other API usage pattern mining techniques. We also would like to help developers to show how many API usage patterns are there in their developing project and whether the uncommon API usage patterns are long-life or not.

## ACKNOWLEDGMENTS

This work has been supported by JSPS KAKENHI Nos. JP18H04094 and JP19K20239.

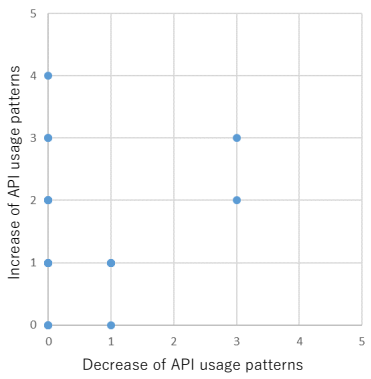


**Figure 10: Increase and Decrease of `Context` API usage patterns**

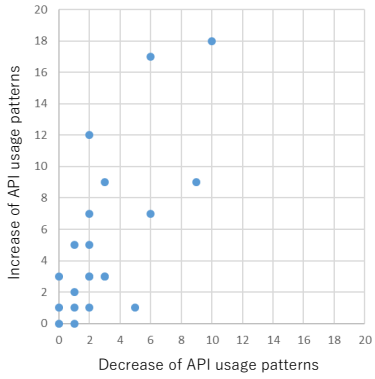


**Figure 11: Increase and Decrease of `Intent` API usage patterns**

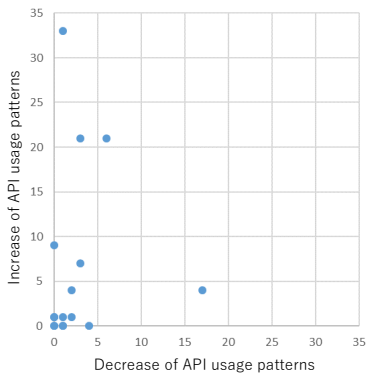


**Figure 12: Increase and Decrease of `Cursor` API usage patterns**

# REFERENCES


[1] Sven Amann, Hoan Anh Nguyen, Sarah Nadi, Tien N. Nguyen, and Mira Mezini. 2019. Investigating next Steps in Static API-Misuse Detection. In *Proceedings of the 16th International Conference on Mining Software Repositories (MSR '19)*. 265–275. https://doi.org/10.1109/MSR.2019.00053

[2] Shams Azad, Peter C. Rigby, and Latifa Guerrouj. 2017. Generating API Call Rules from Version History and Stack Overflow Posts. *ACM Trans. Softw. Eng. Methodol.* 25, 4, Article Article 29 (Jan. 2017), 22 pages. https://doi.org/10.1145/2990497

[3] John Businge, Moses Openja, David Kavaler, Engineer Bainomugisha, Foutse Khomh, and Vladimir Filkov. 2019. Studying Android App Popularity by Cross-Linking GitHub and Google Play Store. In *2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER 2019)*. 287–297. https://doi.org/10.1109/SANER.2019.8667998

[4] Google Developers. [n.d.]. *Android API Reference*. Retrieved February 5, 2020 from https://developer.android.com/reference

[5] Manuel Egele, David Brumley, Yanick Fratantonio, and Christopher Kruegel. 2013. An Empirical Study of Cryptographic Misuse in Android Applications. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security (CCS '13)*. 73–84. https://doi.org/10.1145/2508859.2516693

[6] Jaroslav Fowkes and Charles Sutton. 2016. Parameter-Free Probabilistic API Mining across GitHub. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2016)*. 254–265. https://doi.org/10.1145/2950290.2950319

[7] Google. [n.d.]. *Android*. Retrieved February 5, 2020 from https://www.android.com/

[8] Daqing Hou and Lin Li. 2011. Obstacles in Using Frameworks and APIs: An Exploratory Study of Programmers' Newsgroup Discussions. In *2011 IEEE 19th International Conference on Program Comprehension*. 91–100. https://doi.org/10.1109/ICPC.2011.21

[9] Dino Konstantopoulos, John Marien, Mike Pinkerton, and Eric Braude. 2009. Best Principles in the Design of Shared Software. In *2009 33rd Annual IEEE International Computer Software and Applications Conference*, Vol. 2. 287–292. https://doi.org/10.1109/COMPSAC.2009.151

[10] Zhenmin Li and Yuanyuan Zhou. 2005. PR-Miner: Automatically Extracting Implicit Programming Rules and Detecting Violations in Large Software Code. *SIGSOFT Softw. Eng. Notes* 30, 5 (Sept. 2005), 306–315. https://doi.org/10.1145/1095430.1081755

[11] Mario Linares-Vásquez, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, and Denys Poshyvanyk. 2014. 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)*. 83–94. https://doi.org/10.1145/2597008.2597155

[12] Tyler McDonnell, Baishakhi Ray, and Miryung Kim. 2013. An Empirical Study of API Stability and Adoption in the Android Ecosystem. In *2013 IEEE International Conference on Software Maintenance*. 70–79. https://doi.org/10.1109/ICSM.2013.18

[13] Martin Monperrus and Mira Mezini. 2013. Detecting Missing Method Calls as Violations of the Majority Rule. *ACM Trans. Softw. Eng. Methodol.* 22, 1, Article Article 7 (March 2013), 25 pages. https://doi.org/10.1145/2430536.2430541

[14] Simon Moser and Oscar Nierstrasz. 1996. The effect of object-oriented frameworks on developer productivity. *Computer* 29, 9 (Sep. 1996), 45–51. https://doi.org/10.1109/2.536783

[15] Tung Thanh Nguyen, Hoan Anh Nguyen, Nam H. Pham, Jafar M. Al-Kofahi, and Tien N. Nguyen. 2009. Graph-Based Mining of Multiple Object Usage Patterns. In *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering (ESEC/FSE '09)*. 383–392. https://doi.org/10.1145/1595696.1595767

[16] Martin P. Robillard. 2009. What Makes APIs Hard to Learn? Answers from Developers. *IEEE Software* 26, 6 (Nov 2009), 27–34. https://doi.org/10.1109/MS.2009.193

[17] Martin P. Robillard and Robert DeLine. 2011. A field study of API learning obstacles. *Empirical Software Engineering* 16, 6 (01 Dec 2011), 703–732. https://doi.org/10.1007/s10664-010-9150-8

[18] Caitlin Sadowski, Kathryn T. Stolee, and Sebastian Elbaum. 2015. How Developers Search for Code: A Case Study. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE 2015)*. 191–201. https://doi.org/10.1145/2786805.2786855

[19] Mohamed Aymen Saied, Omar Benomar, Hani Abdeen, and Houari Sahraoui. 2015. Mining Multi-level API Usage Patterns. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. 23–32. https://doi.org/10.1109/SANER.2015.7081812

[20] Mohamed Aymen Saied, Ali Ouni, Houari Sahraoui, Raula Gaikovina Kula, Katsuro Inoue, and David Lo. 2018. Improving reusability of software libraries through usage pattern mining. *Journal of Systems and Software* 145 (2018), 164 – 179. https://doi.org/10.1016/j.jss.2018.08.032

[21] Ferdian Thung, David Lo, and Julia Lawall. 2013. Automated library recommendation. In *2013 20th Working Conference on Reverse Engineering*. 182–191. https://doi.org/10.1109/WCRE.2013.6671293

[22] Miryung Kim Tyler McDonnell, Baishakhi Ray. 2013. An Empirical Study of API Stability and Adoption in the Android Ecosystem. In *Proceedings of the 2013 IEEE International Conference on Software Maintenance (ICSM 2013)*. 70–79. https://doi.org/10.1109/ICSM.2013.18

[23] Medha Umarji, Susan Elliott Sim, and Crista Lopes. 2008. Archetypal Internet-Scale Source Code Searching. In *Open Source Development, Communities and Quality*. Springer US, Boston, MA, 257–263.

[24] Jue Wang, Yingnong Dang, Hongyu Zhang, Kai Chen, Tao Xie, and Dongmei Zhang. 2013. Mining succinct and high-coverage API usage patterns from source code. In *2013 10th Working Conference on Mining Software Repositories (MSR)*. 319–328. https://doi.org/10.1109/MSR.2013.6624045

[25] Wang Wei and Michael W. Godfrey. 2013. Detecting API usage obstacles: A study of iOS and Android developer questions. In *2013 10th Working Conference on Mining Software Repositories (MSR)*. 61–64. https://doi.org/10.1109/MSR.2013.6624006

[26] Ming Wen, Yepang Liu, Rongxin Wu, Xuan Xie, Shing-Chi Cheung, and Zhendong Su. 2019. Exposing Library API Misuses Via Mutation Analysis. In *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE) (ICSE 2019)*. 866–877. https://doi.org/10.1109/ICSE.2019.00093

[27] Tao Xie and Jian Pei. 2006. MAPO: Mining API Usages from Open Source Repositories. In *Proceedings of the 2006 International Workshop on Mining Software Repositories (MSR 2006)*. 54–57. https://doi.org/10.1145/1137983.1137997

[28] Tianyi Zhang, Ganesha Upadhyaya, Anastasia Reinhardt, Hridesh Rajan, and Miryung Kim. 2018. 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 2018)*. 886–896. https://doi.org/10.1145/3180155.3180260

**Table 7: #use of top-5 API usage patterns and project-specific API usage patterns**

| API | Context | | | | Intent | | | | Cursor | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Project | top-5 (%) | | peoject-specific (%) | | top-5 (%) | | peoject-specific (%) | | top-5 (%) | | peoject-specific (%) | |
| AmazeFileManager | 2 | (40%) | 1 | (20%) | 31 | (65%) | 6 | (12%) | 0 | (0%) | 15 | (65%) |
| AndroidUtilCode | 0 | (0%) | 10 | (100%) | 47 | (58%) | 15 | (18%) | 0 | (0%) | 0 | (0%) |
| bitcoin-wallet | 3 | (75%) | 1 | (25%) | 15 | (51%) | 10 | (34%) | 0 | (0%) | 1 | (100%) |
| cgeo | 2 | (50%) | 1 | (25%) | 68 | (72%) | 14 | (14%) | 3 | (20%) | 6 | (40%) |
| codenameone | 20 | (90%) | 1 | (4%) | 26 | (50%) | 15 | (29%) | 0 | (0%) | 6 | (54%) |
| collect | 0 | (0%) | 1 | (25%) | 45 | (54%) | 25 | (30%) | 0 | (0%) | 55 | (96%) |
| ExoPlayer | 2 | (66%) | 0 | (0%) | 4 | (36%) | 5 | (45%) | 0 | (0%) | 0 | (0%) |
| fresco | 7 | (100%) | 0 | (0%) | 4 | (57%) | 1 | (14%) | 0 | (0%) | 2 | (66%) |
| k9 | 37 | (80%) | 5 | (10%) | 77 | (52%) | 45 | (30%) | 6 | (8%) | 49 | (68%) |
| mapbox | 5 | (62%) | 2 | (25%) | 1 | (100%) | 0 | (0%) | 0 | (0%) | 0 | (0%) |
| news-android | 0 | (0%) | 0 | (0%) | 32 | (80%) | 3 | (7%) | 2 | (20%) | 8 | (80%) |
| owncloud | 1 | (33%) | 2 | (66%) | 67 | (68%) | 8 | (8%) | 0 | (0%) | 12 | (80%) |
| owntracks | 1 | (100%) | 0 | (0%) | 5 | (29%) | 2 | (11%) | 0 | (0%) | 2 | (100%) |
| photoview | 0 | (0%) | 0 | (0%) | 1 | (100%) | 0 | (0%) | 0 | (0%) | 0 | (0%) |
| realm-java | 3 | (60%) | 1 | (20%) | 8 | (88%) | 0 | (0%) | 0 | (0%) | 0 | (0%) |
| RedReader | 6 | (60%) | 4 | (40%) | 34 | (52%) | 11 | (16%) | 0 | (0%) | 2 | (40%) |
| robolectric | 0 | (0%) | 0 | (0%) | 0 | (0%) | 1 | (100%) | 0 | (0%) | 0 | (0%) |
| Signal-Android | 20 | (71%) | 4 | (14%) | 102 | (61%) | 31 | (18%) | 9 | (11%) | 51 | (64%) |
| skytube | 0 | (0%) | 1 | (100%) | 23 | (76%) | 2 | (6%) | 0 | (0%) | 17 | (80%) |
| wikipedia | 5 | (45%) | 4 | (36%) | 24 | (51%) | 16 | (34%) | 10 | (55%) | 5 | (27%) |
| WordPress | 28 | (65%) | 9 | (20%) | 140 | (66%) | 31 | (14%) | 22 | (45%) | 40 | (83%) |
| Total | 142 | (65%) | 47 | (21%) | 754 | (61%) | 241 | (19%) | 52 | (13%) | 271 | (71%) |

**Table 8: #pattern for Each Projects**

| API | Context | | Intent | | Cursor | |
|---|---|---|---|---|---|---|
| Project | old | new | old | new | old | new |
| AmazeFileManager | 0 | 4(3) | 15(14) | 17(15) | 7(7) | 9(9) |
| AndroidUtilCode | 0 | 2(2) | 6(6) | 20(16) | 1(1) | 0 |
| bitcoin-wallet | 4(4) | 4(4) | 11(8) | 12(8) | 6(5) | 1(1) |
| cgeo | 1(1) | 3(2) | 26(20) | 28(21) | 10(10) | 12(11) |
| codenameone | 3(3) | 6(4) | 21(18) | 24(19) | 10(9) | 10(9) |
| collect | 1(1) | 2(2) | 14(14) | 23(20) | 9(7) | 44(25) |
| ExoPlayer | 3(3) | 3(3) | 4(4) | 8(5) | 0 | 0 |
| fresco | 0 | 3(2) | 1(1) | 5(4) | 2(2) | 3(3) |
| k9 | 8(5) | 11(7) | 36(33) | 49(33) | 27(24) | 44(39) |
| mapbox | 3(3) | 4(2) | 2(2) | 1(1) | 0 | 0 |
| news-android | 1(1) | 1(1) | 9(8) | 10(9) | 6(6) | 6(6) |
| owncloud | 0 | 2(2) | 26(24) | 25(24) | 8(6) | 9(6) |
| owntracks | 0 | 1(1) | 15(14) | 10(10) | 3(3) | 2(2) |
| photoview | 1(1) | 1(1) | 1(1) | 1(1) | 0 | 0 |
| realm-java | 2(2) | 3(2) | 3(2) | 3(2) | 0 | 0 |
| RedReader | 5(3) | 5(3) | 15(14) | 21(19) | 5(5) | 4(4) |
| robolectric | 1(1) | 0 | 2(2) | 1(1) | 0 | 0 |
| Signal-Android | 8(5) | 10(5) | 31(23) | 44(34) | 16(8) | 56(40) |
| skytube | 0 | 1(1) | 5(4) | 9(8) | 0 | 11(9) |
| wikipedia | 1(1) | 8(5) | 14(9) | 19(12) | 3(3) | 7(7) |
| WordPress | 9(4) | 13(7) | 26(25) | 41(33) | 54(34) | 22(21) |
| Total | 27(15) | 44(18) | 158(123) | 213(144) | 141(96) | 206(136) |

(): w/o instance