API Document Quality for Resolving Deprecated APIs

Deokyoon Ko, Kyeongwook Ma and Sooyong Park
Dept. of Computer Science & Engineering,
Sogang University, Seoul, South Korea
{maniara,shwarzes,sypark}@sogang.ac.kr

Suntae Kim
Dept. of Software Engineering,
Cheonbuk National University,
Jeonju Si, Jeollabuk Do, South Korea
stkim@jbnu.ac.kr

Dongsun Kim and Yves Le Traon
University of Luxembourg, SnT,
Luxembourg
{dongsun.kim,yves.letraon}@uni.lu

***Abstract*—**

Using deprecated APIs often results in security vulnerability or performance degradation. Thus, invocations to deprecated APIs should be immediately replaced by alternative APIs. To resolve deprecated APIs, most developers rely on API documents provided by service API libraries. However, the documents often do not have sufficient information. This makes many deprecated API usages remain unresolved, which leads programs to vulnerable states. This paper reports a result of studying document quality for deprecated APIs. We first collected 260 deprecated APIs of eight Java libraries as well as the corresponding API documents. These documents were manually investigated to figure out whether it provides alternative APIs, rationales, or examples. Then, we examined 2,126 API usages in 249 client applications and figured out whether those were resolved in the subsequent versions. This study revealed that 1) 3.6 APIs was deprecated and 3.6 deprecated APIs are removed from the library a month on average, 2) only 61% of API documents provided alternative APIs while rationale and examples were rarely documented, and 3) 62% of deprecate API usages in client applications were resolved if the corresponding API documents provided alternative APIs while 49% were resolved when the documents provided no alternative APIs. Based on these results, we draw future directions to encourage resolving deprecated APIs.

I. Introduction

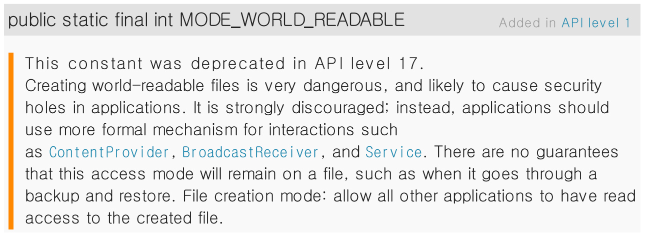
Software developers use several libraries to boost development productivity when writing their applications. This establishes dependencies between an application and libraries. Thus, when the APIs of a library is changed, the application using the library should be also updated accordingly [8].

Some APIs are very frequently updated. By examining Android release notes, 115 APIs are changed per month from release 1.0 to 4.2 [2]. According to [5], API changes occur due to the following reasons:

- Defects: security vulnerability and performance issues,
- Coding Style: it may result in bad coding practices.

This implies that the use of the old API may raise reliability and maintenance issues. Figure 1 shows an example of deprecated APIs due to security vulnerability.

For Java programs, developers often annotate '`@Deprecated`' on each code element (e.g., classes or methods) if it is necessary to suggest stopping the use of the code element. This give a chance for developers to change their applications before permanently removing the deprecated APIs [5], [7]. In addition, sufficient information for the deprecated API should be provided for the library users to update their applications. Otherwise, the API users cannot update their applications according to the new library due to insufficient documentation [1]. In particular, API documents are known as one of the most effective methods to resolve deprecated API since it can convey the intentions why API developers deprecated the APIs [6].


public static final int MODE_WORLD_READABLE   Added in API level 1

This constant was deprecated in API level 17.
Creating world-readable files is very dangerous, and likely to cause security holes in applications. It is strongly discouraged; instead, applications should use more formal mechanism for interactions such as `ContentProvider`, `BroadcastReceiver`, and `Service`. There are no guarantees that this access mode will remain on a file, such as when it goes through a backup and restore. File creation mode: allow all other applications to have read access to the created file.

Fig. 1: Example of deprecated APIs due to security vulnerability.

In this paper, we hypothesize that the quality of API documents may affect the resolution of deprecated APIs. We define the quality of API documentation with respect to deprecated APIs. This illustrates what kind of information is provided for application developers such as alternative APIs, code examples, and rationale to help the resolution of deprecated API.

Three research questions are defined to examine our assumptions as follows:

- *RQ1. How frequently APIs are deprecated?*
- *RQ2. How sufficient information is provided for the deprecated API?*
- *RQ3. How does API document quality have an influence on the resolution of deprecated APIs?*

To obtain an answer for each research question, we built a tool support, *D-API Miner*, to automatically identify newly introduced/resolved/unresolved deprecated APIs. By using the tool, we have investigated 260 deprecated APIs (including classes, methods, and attributes) of eight Java libraries with 48 releases, and also analyzed 2,126 usages of the deprecated APIs in 249 applications to recognize the causal relationship between the quality of API documentation and the resolution of the deprecated API.

TABLE I: Subjects for our study (Dur.=Duration in month, Rel=Releases).

Subject	Description	Versions	Dur.	# of Rel.
Guava	Collection Framework	10 – 17	31	7
Lucene	Search,Index	1.4.3 – 4.8.1	84	42
Hadoop	Distributed file system	1.0.3 – 1.2.1	9	6
Spring	Application Framework	2.5.6 – 4.0.0	19	25
Jetty	Web server and servlet	7.6.14 – 9.0.0	8	23
ActiveMQ	Messaging service	5.6 – 5.9	24	3
Accumulo	Column based database	1.3.0 – 1.6.0	23	9
Camel	SOA message middleware	1.6.4 – 2.13.0	40	48

This investigation reveals that 1) 3.6 APIs of the libraries were monthly deprecated on average. 2) Only 61% of API documents provide alternative APIs for deprecated APIs, while examples for the alternative APIs and the rationale for the deprecation were rarely provided. 3) 62% of deprecated API usages in applications were resolved when alternative APIs were provided in the API document. However, only 39% of deprecated APIs were resolved in the applications when the alternative APIs are not introduced in the document.

This paper is structured as follows: Section II illustrates the further details of the research questions and the corresponding study methods. Section III shows the results of the study as an answer of each research question. After discussing related work on API changes in Section IV, we conclude this paper in Section V.

II. Research Questions & Methods

This section describes our research questions on the documentation quality of deprecated APIs and experiment design to handle each question.

RQ1: How frequently APIs are deprecated?

Before discussing the correlation between the document quality and the deprecated APIs, it is necessary to figure out the frequency of API deprecation. Although few of previous studies [2], [4] reported the frequency of API changes in *java.awt*, *javax.swing* and *Android*, investigating the number of deprecated APIs in our subjects is prerequisite.

We investigate eight Java libraries as shown in Table I. These are selected because they are popular open-source projects and provide publicly available source code. *Versions* in the table indicates an version interval used for our study, and the *Duration* column represents the number of months for the interval. *# of Releases* shows the number of individual major or minor version releases used for our study.

RQ2: How sufficient information is provided for the deprecated API?

This question addresses which aspect defines the quality of API documents for deprecated APIs. In this study, we manually investigate the documentation quality of the subjects shown in Table I with respect to the following aspects:

- Does the API document provide alternative APIs for deprecated APIs?
- Does the API document include any example for the alternative APIs?
- Does the API document provide a detailed guide to resolve deprecated APIs by using an alternative API?
- Does the API document present the reason of the deprecation?

TABLE II: Client applications examined in our study. For each library API, we collected several client applications to answer our research questions.

API Library	Client Applications
Guava	Jcbi-xml, Accumulo-core, Jcloud-component core
Lucene-core	Jackrabbit-core, Camel lucene, Elasticsearch, Solr-core
Hadoop-core	Hadoop-mapreduce, Nutch, Camel-hbase, Hadoop-NFS
Spring-core	Spring-AOP, Spring-beans, Spring-context, Spring-JDBC
Jetty-server	Jetty-Asynchronous HTTP client, Jetty-SPDY, Jetty-test webapp
ActiveMQ-core	OpenEJB-container core, OpenEJB-TomEE, ServiceMix-core
Accumulo-core	Accumulo-wikisearch query, Accumulo-server base
Camel-core	Camel-FTP, Camel-EJB, Camel-geocoder, Camel-gson

TABLE III: Monthly average number of added/removed deprecated APIs (numbers in parentheses are standard deviation).

Libraries	New	Removed	Sum	Total	First Release
Guava	5.8	3.7	9.5	295	Sep. 2009
Lucene-core	1.3	0.5	1.8	150	Oct. 2000
Hadoop-core	0.2	0.0	0.2	2	Mar. 2012
Spring-core	1.5	6.1	7.6	144	Dec. 2005
Jetty-server	2.5	6.5	9.0	72	Apr. 2009
ActiveMQ-core	0.8	0.6	1.5	35	Nov. 2005
Accumulo-core	11.9	9.2	21.1	486	Jul. 2012
Camel-core	4.6	2.0	6.6	263	Jun. 2007
Sum(standard deviation)	3.6 (3.9)	3.6 (3.4)	7.2	1447	

RQ3: How does API document quality have an influence on the resolution of deprecated APIs?

This research question examines the correlation between the quality of API documentation and the resolution of deprecated APIs. We analyze 249 pairs of library APIs (listed in Table I) and client applications for these research questions. For each library APIs, we collected several client applications as shown in Table II.

III. Results

RQ1: How frequently APIs are deprecated?

We investigated the API documents of the eight libraries with 163 releases and summarized the results in Table III. In the table, the *New* and *Removed* columns are the number of newly introduced/removed deprecated APIs from the previous on average, respectively. We compared two adjacent versions of each subject library to count the *New* and *Removed* deprecated APIs. If a new deprecated API is discovered after comparing with the previous release, we counted it as a newly introduced deprecated API. Thus, deprecated APIs already contained in the previous release were ignored. In addition, when the API is discovered in the previous release but not found in the next release, we counted it as a removed deprecated API. At last, all discoveries are averaged by dividing it by months (*Duration*) to see its trend. Projecting the result in monthly average is valuable because the interval between releases of a library is considerably different each other.

As shown in Table III, on average, 3.6 APIs with 3.9 standard deviation become newly deprecated and 3.6 APIs with 3.4 standard deviation are removed every month. After examining the causes of the difference, we figured out it is strongly related to the first release date of each library. According to the table, the number of the deprecated API is getting smaller as the library is released on the earlier date. Therefore, *Lucene-core* and *ActiveMQ-core* has the least number of the deprecated API, which can be considered as the most stable. On the other hand, *Accumulo-core*, *Guava* and *Jetty-server* have relatively high deprecated APIs. Interestingly,

TABLE IV: Amount of provided information of newly added deprecated API (Alt=Alternative, Smpl=Sample Code, Migr=Migration Guide, Rsn=Reason).

Libraries	New	Alt	Smpl	Migr	Rsn	Alt/New	Rsn/New
Guava	180	110	0	0	18	0.61	0.10
Lucene-core	110	57	0	0	5	0.52	0.05
Hadoop-core	2	2	0	0	0	1.00	0.00
Spring-core	29	2	0	0	0	0.07	0.00
Jetty-server	20	13	0	0	0	0.65	0.00
ActiveMQ-core	20	17	0	0	2	0.85	0.10
Accumulo-core	274	219	1	0	16	0.80	0.06
Camel-core	185	80	0	0	2	0.43	0.01
	820	500	1	0	43	0.61	0.05

Hadoop-core has very smaller number of deprecation though it is recently released. It is because it is almost not newly released after the first release. The result can be also interpreted that the library APIs may generate 7.2 causes every month that the applications should be updated according to the depreciated APIs.

RQ2: How sufficient information is provided for the deprecated API?

In order to address this question, we analyzed the content of the API documents for the newly introduced deprecated APIs in the previous research question (see the *New* column in Table III). As the removed API cannot be found in the recent API document, it is excluded for analyzing the quality of the API documents. For the newly introduced deprecated APIs, we counted the number of alternative APIs, sample codes, migration guidelines and deprecation reason of them.

Table IV shows the results of the API document analysis. The first *New* column indicates the total number of deprecated APIs discovered in the RQ1 (e.g., 5.8×31 months = $179.8 \approx 180$ for *Guava*). When the document provides the alternative classes, methods or attributes for the deprecation, we counted it as an *Alternative*. Others are similarly measured by checking if the item exists in the API documents, we counted it. In addition, the proportion of the *Alternative* and *Reason* of all newly introduced deprecated APIs are computed at the last two columns.

The result shows that 61% of API documents for the deprecated APIs offered at least one alternative API. This indicates that the developers must figure out appropriate alternative APIs for the 39% deprecated APIs without the support of API documents. Unfortunately, most of the API documents do not provide a sample code and a migration guide for deprecated APIs as shown in Table IV. We guess that the reason is caused by the fact that the Java API specification, which is the origin of API specification of Java world, does not provide sample codes or guidelines, and many API documents adopt their documentation styles from the Java API specification.

RQ3. How much does API document quality have an influence on application's code update?

To answer this research question, we collected pairs of an API library and a client application that are co-changed simultaneously as shown in Figure 2(a). For every version-up of a client application and library, there might have deprecated API usages. In the new client application, there would have three kinds of deprecated API usages which are generated from old library, new library, or added functionality (Figure 2(b)).

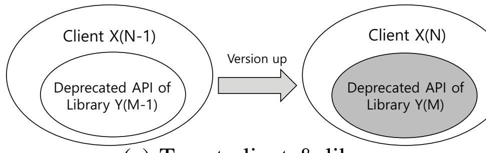
TABLE V: Impact of existence of alternative APIs in client applications. The number of deprecated APIs without duplicates are shown in parentheses.

Libraries	generated API usage	resolved API usage	unresolved API usage
Guava	4 (4)	0 (0)	4 (4)
Lucene-core	119 (74)	48 (36)	71 (38)
Hadoop-common	3 (0)	1 (0)	2 (0)
Hadoop-core	1 (1)	0 (0)	1 (1)
Spring-core	13 (4)	10 (3)	3 (1)
Jetty-server	1 (0)	1 (0)	0 (0)
ActiveMQ-core	1 (1)	0 (0)	1 (1)
Accumulo-core	76 (64)	58 (50)	18 (13)
Camel-core	42 (21)	22 (15)	20 (6)
	260 (169)	140 (104)	120 (64)

We consider the deprecated API usage of second one as indicated by the grey part of Figure 2(b).

Table V represents the impact of the existence of alternative APIs to client applications. The other factor such as *Reason* are not shown in the table as no significant correlation is available for them. The *generated deprecated API usage* (the second column) is the number of new introduced deprecated APIs usages when a client developer upgrades to a new version of an API library. The *resolved deprecated API usage* (the third column) is the number of API usages removed in a new version of client applications. This implies that client developers resolved the deprecated APIs. *Unresolved deprecated API usage* (the fourth column) is the number of remaining deprecated on a new version of client applications.

We counted the number of unique deprecated APIs, which implies that no duplicate was allowed. In addition, we decide the unresolved deprecated APIs if any of the unique deprecated APIs remaining in a new version of client applications. We found 260 generated deprecated APIs when their API libraries were changed and 120 deprecated APIs remained unresolved. The numbers in parentheses represent the number of APIs that has alternative API. Figure 3 shows the ratio of the study result. Figure 3(a) indicates that 75% of resolved APIs have alternative APIs, whereas unresolved APIs have 53%. Figure 3(b) indicates 62% resolved in client applications among deprecated APIs having an alternative API. On the other hand, only 39% of non-alternative APIs. This survey shows us the importance of the existence of alternative API. Alternative API is one of the essential aids of changing their code.


Client X(N-1)
Deprecated API of Library Y(M-1)
Version up
Client X(N)
Deprecated API of Library Y(M)
(a) Target client & library

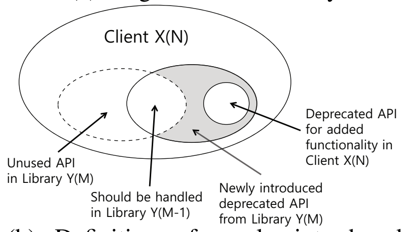
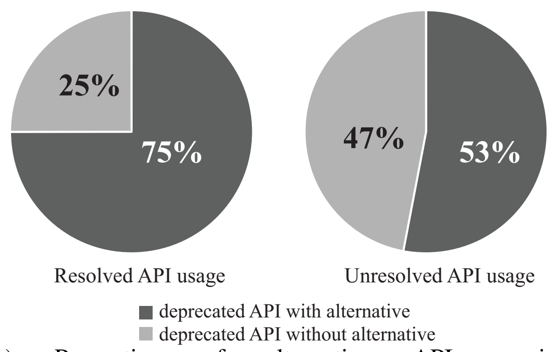
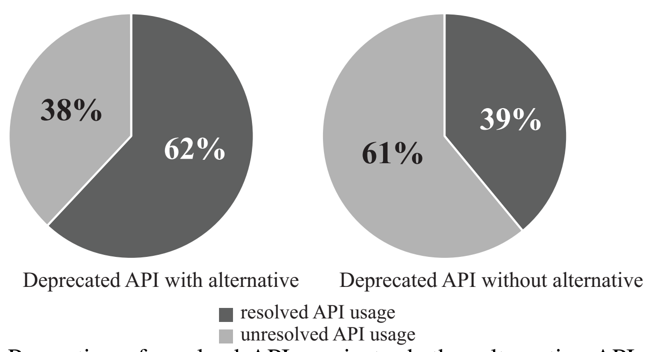

Client X(N)
Deprecated API for added functionality in Client X(N)
Unused API in Library Y(M)
Should be handled in Library Y(M-1)
Newly introduced deprecated API from Library Y(M)
(b) Definition of newly introduced Deprecated API

Fig. 2: Study method for RQ3.



(a) Proportion of alternative APIs against resolved/unresolved deprecated APIs.



(b) Proportion of resolved APIs against whether alternative APIs are available or not.

Fig. 3: Comparison the proportion of resolved and unresolved APIs with respect to whether alternative APIs are available or not.

IV. Related Work

Other researchers have conducted surveys on side effect in accordance with API evolution. McDonnell *et al.* [4] analyzed the change pattern of android API and they investigated change period and increasing rate of the number of bugs in client application used android API. According to this paper, client developers have taken 36 weeks to apply 100% of the new API and API that change more frequently causes more bugs. Espinha *et al.* [1] performed a study to identify the reason why the web client library is rarely changed. According to this research, the client developers do not change their code when they update library due to poor documentation, no evolution standard, short deprecated period. Because of those reasons, they argue that the changing standards of web library APIs required. Linares-Vasquez *et al.* [3] reported on a study of mobile based API change pattern and how often the discussion of the API used in mobile software occurred on Stackoverflow depends on their change pattern. This paper shows that frequently changing APIs make developer actively discussed and *deleting public method* generates more relevant and discussed question in the developer community.

V. Conclusion

In this paper, we assumed that the quality of API documents may affect the resolution of deprecated APIs. Then we set up three questions about our hypothesis. To obtain the answers of the three questions, we selected nine Java based libraries. First, we investigated the variation of deprecated APIs. Through the investigation, we found that 7.2 deprecated APIs are added and removed on average every month. Second, we counted the number of supporting information for deprecation. According to the result, API documents revealed 61% of deprecated APIs offer alternative code and only 5% of APIs explained the reason why these APIs are deprecated. Additionally, most of APIs do not provide sample code and evolution example. Third, we classified the client code using deprecated API as existence of alternatives and we counted the resolution rate in each case. Through our survey, if an alternative API is available in API documents, 62% of deprecated API usages were resolved, while only 39% of deprecated API usages were resolved if there is no alternative API supported. More 20% of resolution rate improvement occurred on the basis of the presence of the alternative API. Our findings can inform when the APIs are changed, alternative API will be significantly helpful for changing client applications.

Based on the results reported in this paper, we can conclude that API developers often deprecate APIs without sufficient information. However, deprecated API usages are more likely to be resolved once alternative APIs are provided in API documents. This encourages API developers to specify further details such as alternative APIs when they mark deprecated APIs. In addition, we can design a technique to automate the resolution of deprecated APIs if alternative APIs are available in API documents.

VI. Acknowledgement

This research was supported by Next-Generation Information Computing Development Program through the National Research Foundation of Korea(NRF) funded by the Ministry of Science, ICT & Future Planning (NRF-2014M3C4A7030505)

References

[1] T. Espinha, A. Zaidman, and H.-G. Gross. Web api growing pains: Stories from client developers and their code. In *proceedings of the 2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering and Reverse Engineering (CSMR-WCRE)*, pages 84–93, Feb 2014.

[2] D. Hou and X. Yao. Exploring the intent behind api evolution: A case study. In *Proceedings of the 2011 18th Working Conference on Reverse Engineering*, pages 131–140, Oct 2011.

[3] M. Linares-Vásquez, G. Bavota, M. Di Penta, R. Oliveto, and D. Poshyvanyk. How do api changes trigger stack overflow discussions? a study on the android sdk. In *proceedings of the 22nd International Conference on Program Comprehension*, pages 83–94. ACM, 2014.

[4] T. McDonnell, B. Ray, and Miryung Kim. An empirical study of api stability and adoption in the android ecosystem. In *proceedings of the 2013 29th IEEE International Conference on Software Maintenance (ICSM)*, pages 70–79, Sept 2013.

[5] Oracle. How and when to deprecate apis, June 2014.

[6] C. Parnin and C. Treude. Measuring api documentation on the web. In *Proceedings of the 2nd International Workshop on Web 2.0 for Software Engineering*, Web2SE '11, pages 25–30, New York, NY, USA, 2011. ACM.

[7] J. H. Perkins. Automatically generating refactorings to support api evolution. In *proceedings of the 6th ACM SIGPLAN-SIGSOFT Workshop on Program Analysis for Software Tools and Engineering*, PASTE '05, pages 111–114, New York, NY, USA, 2005. ACM.

[8] Z. Xing and E. Stroulia. Api-evolution support with diff-catchup. *IEEE Transactions on Software Engineering*, 33(12):818–836, Dec 2007.