# A study on deprecated APIs

Mahta Ghorbanian

UCID: 30126039

SENG 641

2022-04-28

# Introduction and Problem

Application Programming Interfaces (API) helps developers in the development process to provide reusable functionality [1]. Over time the API of a framework or library changes, its developers must be found and fixed them. Deprecation is a language feature that exists to give API producers a way in which they can show that a feature should no longer be used [2]. According to the official Java documentation: "A deprecated API is one that you are no longer recommended to use, due to changes in the API. While deprecated classes, methods, and fields are still implemented, they may be removed in future implementations, so you should not use them in new code, and if possible, rewrite old code not to use them." [3]. The Java language designers noticed that no one was taking these deprecation warnings seriously and continued using deprecated features [1]. Thus, deprecated APIs are a challenge for developers and there are some published works related to this topic.

In this study, I implemented a process that investigates the evolution of deprecated API usage, especially on methods and changes in java projects automatically. The case study is on 40 java projects with all of their version from Maven repository that use a list of predefined libraries. This implementation finds deprecated APIs in each project and then I analyzed the output data.

This study helps us to understand the evolution of deprecated APIs and find out how much the developers pay attention to the deprecation warning. Also, we would understand that are programs UpToDate to these kinds of changes or not? It would give us an overview of the developer's behavior on maintaining their projects and the results could be important for security analysts and the most important usage that this research would have is that the companies or teams could use this research and review the evolution of their projects and see if they do successfully on handling depredated and outdated APIs or not? They just need to input their used libraries by POM file of their project and the project to the app and get the database and analyze the data. (A Project Object Model or POM is the fundamental unit of work in Maven. It is an XML file that contains information about the project and configuration details used by Maven to build the project.[5])

# Procedure and Raw Data and Research Questions

This project has 3 main sections. This project is used 40 java projects and 15 java libraries from Maven repository. These are 40 and 15 because of the limited time.

**Section 1:** Extracting deprecated APIs from all versions of selected libraries

**Section 2:** Extracting usage of deprecated APIs from all versions of selected projects

**Section 3:** Analysis of the extracted data

### Section 1 (Raw Data 1):

In this section, I prepared 15 java libraries link from Maven Repository and give them to a python code. This code goes to each link and downloads all version of that library and get its source code as a jar file. Then all of the jar files extract and each version of libraries would be put in a folder with this format: nameOfLibrary_Year_Version. These data are stored from the libraries link at same time with downloading the jar files. Bellow picture shows a part of the libraries' files. For example, the first file shows "cglib" library which is released on 2005 and its version is 2.1_2. (613 version are extracted from these 15 libraries in total.)

| | | |
|---|---|---|
| cglib_2005_2.1_2 | 2022-04-23 4:53 PM | File folder |
| cglib_2005_2.1_3 | 2022-04-23 4:53 PM | File folder |
| cglib_2008_2.2 | 2022-04-23 4:53 PM | File folder |
| cglib_2011_2.2.2 | 2022-04-23 4:53 PM | File folder |
| cglib_2013_3.0 | 2022-04-23 4:53 PM | File folder |

After I prepared these libraries' files, in a java code, I read each file and all java file and by helping JavaParser parsed them and extract all methods decelerations (The JavaParser is a library that provides Abstract Syntax Tree of your Java code. The AST structure then allows you to work with your Java code.[4]). The parser visits each method and I would have access to name of method, parameters, annotations and exc. I got methods annotations. If the method annotation contains "@deprecated" store the name of method, arguments, return type of it. (I couldn't collect deprecated classes because lack of time ☹)

For having a Table in my database like bellow:

| | 1.2.2 | 1.2.3 | 1.3 | 1.3.1 |
|---|---|---|---|---|
| M() | dep | undep | dep | del |
| N() | dep | dep | dep | dep |
| O() | dep | del | del | del |

I did some process on stored data and then save it into my database. Each method in the table could have 3 options. "dep" or deprecated which means in a version that method has "@deprecated" annotation, "undep" or undeprecated which means that method in previous versions had "@deprecated" annotation but now it doesn't have and "del" or delete means that method is deleted. For having these data in my table, I define "status" column in my table. The code would start from the first file. Because it is the first library and first version of it, the code would add all of methods with "@deprecated" annotation and set "status" as "dep". For next files, again the code would add all of methods with "@deprecated" annotation and set "status" as "dep" and also for each method that is stored in previous step if it doesn't find in this step, set its "status" as "dep" and if the method found and it doesn't have "@deprecated" annotation, set its "status" as "undep". The code will do it until the end for each library and store data like name of library, version, year, name of method, parameters, status and so on, on "libraries_data" table. The bellow picture shows a part of the table. Also whole of the table is attached to the report by name of "sql-export.sql". For example, the firdt row shows in cglib v3.2.1 "singleton" method has "@deprecated" annotation. These table has 167604 rows.

| | id | library | version | Date | return_type | name | parameter | signature | parameter_number | status |
|---|---|---|---|---|---|---|---|---|---|---|
| Edit Copy Delete | 1 | cglib | 3.2.1 | 2016 | CustomizerRegistry | singleton | [Customizer customizer] | CustomizerRegistry singleton [Customizer customize... | 1 | dep |
| Edit Copy Delete | 2 | cglib | 3.2.1 | 2016 | void | hash_code | [CodeEmitter e, Type type, int multiplier, final C... | void hash_code [CodeEmitter e, Type type, int mult... | 4 | dep |
| Edit Copy Delete | 3 | cglib | 3.2.1 | 2016 | void | not_equals | [CodeEmitter e, Type type, final Label notEquals, ... | void not_equals [CodeEmitter e, Type type, final L... | 4 | dep |
| Edit Copy Delete | 4 | cglib | 3.2.1 | 2016 | void | append_string | [final CodeEmitter e, Type type, final ArrayDelimi... | void append_string [final CodeEmitter e, Type type... | 4 | dep |
| Edit Copy Delete | 5 | cglib | 3.2.1 | 2016 | void | setCustomizer | [Customizer customizer] | void setCustomizer [Customizer customizer] | 1 | dep |
| Edit Copy Delete | 6 | cglib | 3.2.1 | 2016 | CustomizerRegistry | singleton | [Customizer customizer] | CustomizerRegistry singleton [Customizer customize... | 1 | dep |

*Figure 1 Raw Data 1 "libraries_data" table*

## Section 2 (Raw Data 2):

In this section, I selected 40 java projects link randomly from Maven Repository and give them to a python code. Again, this code goes to each link and downloads all version of that project and get its source code as a jar file. Then all of the jar files extract and each version of projects would be put in a folder with this format: nameOfProject_Year_Version. These data are stored from the projects link at same time with downloading the jar files. Bellow picture shows a part of the projects' files. For example, the first file shows "aliyun-java-sdk-core" project which is released on 2015 and its version is 2.1.0. (2957 version are extracted from these 40 projects in total.)

| | | |
|---|---|---|
| aliyun-java-sdk-core_2015_2.1.0 | 2022-04-24 2:16 AM | File folder |
| aliyun-java-sdk-core_2015_2.1.1 | 2022-04-24 2:16 AM | File folder |
| aliyun-java-sdk-core_2015_2.1.2 | 2022-04-24 2:16 AM | File folder |
| aliyun-java-sdk-core_2015_2.1.3 | 2022-04-24 2:16 AM | File folder |

Also, the code extracts pom files of each version of projects to find that each project use which version of libraries. Pom files used xml language. I used a xml parser in java to extract versions and libraries of pom files. Bellow picture shows a part of pom file. At first, I get “dependencies” tag then for each “dependency” tag, I store the “artifactId” and “version”.

```
<dependencies>
  <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.12</version>
      <scope>test</scope>
  </dependency>
</dependencies>
```

Now, for each version of projects based on data that we have from pom file for that version of project (libraries and versions) query on the “libraries_data” table and get all of the deprecated methods that those particular versions of libraries have as a “deprecated_methods” list. For example, bellow picture is a result of the query for project X. Based on pom file of project X uses “commons_lang3” version 3.0.1 and “slf4j_api” version 1.8.0_alpha1. And these 5 methods are deprecated methods that are in these libraries. And if project X uses these methods, it means they use deprecated method in their project.

| name | parameter_number | library | version |
|---|---|---|---|
| getDefaultCauseMethodNames | 0 | commons-lang3 | 3.0.1 |
| getCause | 1 | commons-lang3 | 3.0.1 |
| getCause | 2 | commons-lang3 | 3.0.1 |
| toCharacterObject | 1 | commons-lang3 | 3.0.1 |
| hasChildren | 0 | slf4j-api | 1.8.0-alpha1 |

The next step is parsing projects java files. Again, I used JavaParser and visit each method calls and if they have same name and same parameter number with methods in “deprecated_methods” list from previous step, it means they used a deprecated methods in their code (I used number of parameters because JavaParser doesn’t get the type of parameters correctly). And the data like name of method, number of parameters, name of project, version of project, realize year of project , library name and library version would store in the “project_Data” table. This process was done for all of the projects. Based on row numbers of this table, in all projects 6373 deprecated methods are used. Bellow picture shows a part of “sql-export.sql” table. For example, in “jcabi_http” project version 1.14.1, they used “getSocketFactory” method which was a deprecated method in “httpclient” library versio 4.5.1. whole of the table is attached to the report by name of “Projects_Data.sql”.

| id | project_name | project_version | project_year | method_name | par_num | library_name | library_version |
|---|---|---|---|---|---|---|---|
| 823 | jcabi-http | 1.14.1 | 2015 | getSocketFactory | 0 | httpclient | 4.5.1 |
| 824 | jcabi-http | 1.14.2 | 2016 | getSocketFactory | 0 | httpclient | 4.5.1 |
| 825 | jcabi-http | 1.15 | 2016 | getSocketFactory | 0 | httpclient | 4.5.1 |
| 826 | jcabi-http | 1.16 | 2016 | invalidate | 0 | httpclient | 4.5.1 |

*Figure 2 - Raw Data 2 “project_Data” table*

### Section 3 (Research Questions):

After collecting data finished, we reach to section3 Analyzing the data and answering research questions. I answered these questions by querying the already extracted tables from the database. There are 5 research questions:

**1. Do project owners ever release a version to fix deprecations or do they just leave it? If yes, what portions of it?**

To answer this question, I investigate "project_Data" table. First, I selected all of deprecated methods in a project except the last version then searching these methods in the last version's methods. And calculate what percentage of these deprecated methods were fixed.

**2. On average, how long does it take for projects to fix all of their deprecated APIs?**

By using "project_Data" table, I compared year of first deprecation of a method in the table with year of fixing all of the deprecated methods and get the average.

**3. What percentage of deprecated APIs are fixed in each version?**

The solution for this question is similar to question 1. Instead of comparing last version with all of the previous versions, I compared previous version with next version.

**4. Do fixing a deprecated API have priority for developers?**

If they covered most of the deprecated methods in their next version, it means that it had priority for developers. I used the percentage that was calculated in Q3.

**5. Is there any similar work with my project? If yes, what are the differences and similarities in the result?**

Sawant, A.A., Robbes, R. & Bacchelli, A. To react, or not to react: Patterns of reaction to API deprecation. Empir Software Eng 24, 3824–3870 (2019). This is the only paper that I could find in this subject. I will say about the similarities and differences in the next part.

# Results

From 40 projects based on "project_Data" table, 26 of projects were not exist in the table. It could be because of 3 things: 1. These projects didn't use any deprecated methods at all that we can conclude 65% of projects pay attention to deprecated warning and didn't use them. 2. Their Pom files didn't follow the maven structure or their dependency tags are empty. 3. The libraries that they used are not in our 15 libraries. I had some solution to understand reason of each project (I mentioned it in future work) but because of limited time I couldn't implement it and do all of the process again. Thus, because we have 3 options for these 26 projects, we couldn't conclude anything for sure. But for the rest of the projects, we can answer research questions.

**Q1.** 3 projects didn't fix deprecated methods at all. 11 projects fixed all of their deprecated methods by releasing a version. Totally, we can say 75% of projects at last released a version without using any deprecated method. And 25% never fixed them.

**Q2.** Except 3 projects that didn't fix deprecated methods at all. 8 projects fixed all of their deprecated methods by releasing a version less than one year. 2 projects fixed all of their deprecated methods by releasing a version after one year. And one project fixed all of their deprecated methods by releasing a version after 5 years. Totally, we can say these 11 projects at last released a version without using any deprecated method on average after 1.36 year later.

**Q3.** Mostly number of deprecated methods which is used in previous version and next version doesn't have any change and the percentage of deprecated methods are fixed in each version is zero. But in one version they fix all of them and the percentage is 100%. Thus, developers didn't remove deprecated methods from their projects exactly in their next version but in a version in future they fixed all of them. Maybe we can conclude that in some point they release a version with goal of removing deprecated APIs.

**Q4.** We couldn't say that fixing deprecated APIs has high priority for developers because in the result we couldn't see any project that fix it exactly in its next version. But we could say it is important for developers and they pay attention to fix it at a time.

**Q5.** This paper has same subject with my study but the data that are collected and Research questions are different. In this paper, they manually analyze 380 usages of deprecated API artifacts across consumers of 50 APIs. Based on this analysis, they observed seven reaction patterns to deprecation [1]. And they focused on these reaction patterns. So, as my understanding from this paper, I couldn't find any special thing to compare my results with their results.

# Conclusion

In this project I tried to implement automatic process to investigate deprecated APIs in 40 projects from Maven Repository. And the result of this study based on the data and those 14 projects is that 75% of projects at last released a version without using any deprecated method about one year later. And fixing deprecated APIs doesn't have high priority for developers but we could say mostly they release a version with goal of fixing all deprecated APIs.

# Future Work

In future, I could Get more projects to this project and extend the case study to reach to a more accurate results that we can expand the results to all of the projects in the real world. The next one is solving the problem that I mentioned it in the first part of Results section, change the project to count number of each reason that we can analyze that part of case study too. We also have deprecated classes that I didn't have enough time to implement. In future, I can implement that part too. Also, in table instead of storing year, storing timestamp in order to have more accurate data for dates. Finally, this project has the potential to become a tool that developers or projects owners could get their project with its pom file to this tool and get the raw data to analyze it and see how much they did well in facing to deprecated APIs.

# References


1. Sawant, A.A., Robbes, R. & Bacchelli, A. To react, or not to react: Patterns of reaction to API deprecation. Empir Software Eng 24, 3824–3870 (2019).

2.https://dl.acm.org/doi/abs/10.1145/2393596.2393662?casa_token=EXYI7L2EvboAAAAA:lFwgfx_bCAWAHVKTnX_hFgilk5JRPiKusb_i00YrS-8LiQx_RLMJGaiQbdXFXj8P2eIValc8Z5I

3. https://docs.oracle.com/javase/8/docs/technotes/guides/javadoc/deprecation/index.html

4. https://javaparser.org/

5. https://maven.apache.org/guides/introduction/introduction-to-the-pom.html

6.https://www.sciencedirect.com/science/article/pii/S016412121730300X?casa_token=YdOBGr0HvmcAAAAA:IWE72pq3lWKpIYu1TEhYGXwNUOEw5NeKd-G5ZB81-xFo_3X4RRUvCgggen6fRj_jbfstkf4r

7. https://dl.acm.org/doi/abs/10.1145/2950290.2950298?casa_token=-cE93A2Z6cAAAAAA:WklgbnyWG3ShvC4jBo_tYGrpBY5ZbWY-nGqT30faCv0pnDl2ZHqFTEPW-qijOBbLnkc92VD6udM