Availability and Usage of Platform-Specific APIs: A First Empirical Study

Ricardo Job
IFPB
Cajazeiras, Brazil
ricardo.job@ifpb.edu.br

Andre Hora
Department of Computer Science, UFMG
Belo Horizonte, Brazil
andrehora@dcc.ufmg.br

ABSTRACT

A *platform-specific API* is an API implemented for a particular platform (e.g., operating system), therefore, it may not work on other platforms than the target one. In this paper, we propose a first empirical study to assess the availability and usage of platform-specific APIs. We analyze the platform-specific APIs provided by the Python Standard Library and mine their usage in 100 popular systems. We find that 21% of the Python Standard Library APIs are platform-specific and that 15% of the modules contain at least one. The platforms with the most availability restrictions are WASI (43.69%), Emscripten (43.64%), Unix (6.76%), and Windows (2.12%). Moreover, we find that platform-specific APIs are largely used in Python. We detect over 19K API usages in all 100 projects, in both production (52.6%) and test code (47.4%). We conclude by discussing practical implications for practitioners and researchers.

CCS CONCEPTS

• **Software and its engineering → Software testing and debugging**.

KEYWORDS

software testing, mining software repositories, test smells, Python

ACM Reference Format:
Ricardo Job and Andre Hora. 2024. Availability and Usage of Platform-Specific APIs: A First Empirical Study. In *21st International Conference on Mining Software Repositories (MSR '24), April 15–16, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3643991.3644925

1 INTRODUCTION

Application Programming Interfaces (APIs) offer multiple benefits to users, such as feature reuse, productivity improvement, and reduction of development costs [9, 16, 17, 19]. A *platform-specific API* is an API implemented for a particular platform (e.g., operating system), therefore, it may not work on other platforms than the target one. For example, some APIs provided by the Python Standard Library have availability restrictions, such as the API `os.listvolumes`,[1] which is available only for Windows, and the API `os.chown`,[2] which is is available only for Unix.

Nowadays, software systems are often tested on multiple platforms to increase quality and avoid bugs. This is supported by containers and modern CI/CD tools, such as GitHub Actions, which make it simple to test, for example, in the Linux, Windows, and macOS operating systems [7]. In this context, platform-specific APIs are problematic because they may break test suites on a certain platform if not properly used. Thus, a system that targets multiple platforms but uses platform-specific APIs should implement defensive code to ensure it will properly work on the desired platforms. For instance, Figure 1 presents usage examples of platform-specific APIs in three real-world projects: Tornado, Django, and Ray.

```python
if sys.platform != "win32":
    # Clear the alarm signal set by
    # ioloop.set_blocking_log_threshold so it doesn't fire
    # after the exec.
    signal.setitimer(signal.ITIMER_REAL, 0, 0)
```

(a) Snippet of `_reload` in Tornado.

```python
try:
    os.chown(full_path, uid=-1, gid=location_gid)
except PermissionError:
    pass
```

(b) Snippet of `_ensure_location_group_id` in Django.

```python
if hasattr(os, "pathconf"):
    return os.pathconf("/", "PC_PATH_MAX")
# Windows
return _DEFAULT_WIN_MAX_PATH_LENGTH
```

(c) Snippet of `_get_max_path_length` in Ray.

Figure 1: Usage examples of platform-specific APIs.

Figure 1a shows a call to the platform-specific API `signal.setitimer`[3] inside an if block to ensure that the current operating system is not Windows. Similarly, Figure 1b presents a case in which the platform-specific API `os.chown`[4] is called within a try-except

[1] https://docs.python.org/3/library/os.html#os.listvolumes
[2] https://docs.python.org/3/library/os.html#os.chown
[3] https://github.com/tornadoweb/tornado/blob/f5df43f26bb4d00759176f7cbec8bdce69f2f4f/tornado/autoreload.py#L201
[4] https://github.com/django/django/blob/311718feb5f1fb9ff794bbac0cda48cfc3410de8/django/core/files/storage/filesystem.py#L139

MSR '24, April 15–16, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0587-8/24/04…$15.00
https://doi.org/10.1145/3643991.3644925

block, while Figure 1c shows a call to the platform-specific API `os.pathconf`[5] inside an if block to ensure the API really exists in the current operating system.

Despite the importance of platform-specific APIs to build real-world software projects, we are not aware of their real availability nor their possible impact on client systems. This knowledge can be used to better understand whether developers are aware of the API restrictions and support the creation of novel guidelines for using platform-specific APIs, like commonly adopted defensive practices. While APIs, in general, is a research topic broadly studied by prior literature (e.g., [2, 5, 8, 10–15, 18, 20–25]), to the best of our knowledge, the platform-specific APIs have never been deeply explored by the research community.

In this paper, we propose a first empirical study to assess the platform-specific APIs. Specifically, we explore (RQ1) the availability of platform-specific APIs provided by the Python Standard Library and (RQ2) their usage in 100 real-world systems. We find that 21% of the Python Standard Library APIs are platform-specific and that 15% of the modules contain at least one. The platforms with the most availability restrictions are WASI (43.69%), Emscripten (43.64%), Unix (6.76%), and Windows (2.12%). Moreover, platform-specific APIs are largely used in Python. We detect 19,288 usages of 683 platform-specific APIs in all 100 projects, in both production (52.6%) and test code (47.4%). Finally, we discuss implications for practitioners and researchers. Our results are publicly available [1].

Contributions. The contributions of the paper are threefold. First, we present the first empirical study to analyze the availability of platform-specific APIs. Second, we propose to assess the usage of platform-specific APIs in real-world systems. Third, we provide a set of implications for researchers and practitioners.

2 STUDY DESIGN

2.1 Case Study

We aim to assess real-world and relevant APIs and software systems. We selected Python due to its popularity and the rich software ecosystem with widely adopted projects to support web development, machine learning, and data analysis

We analyze the availability of platform-specific APIs provided by Python Standard Library. This library is fundamental to building any Python application, providing features to handle text processing, file access, persistence, and networking, to name a few.

We also analyze the usage of the platform-specific APIs. For this purpose, we selected the top 100 most popular Python software systems hosted on GitHub sorted in descending order of the number of stars [3, 4]. We relied on the GitHub Search tool (GHS) [6][6] to find the 100 software projects. In this process, we took special care to filter out non-software projects, such as tutorials, examples, and code samples. On the median, the projects have 17,203 stars, 3,710 commits, and 84 test files.

2.2 Detecting Platform-Specific APIs

A *platform-specific API* is an API implemented for a particular platform, therefore, it may not work on other platforms than the target one. For example, the API `os.listdrives`[7] returns a list with the names of drives on Windows systems only.

Next, we describe the steps to (A) collect APIs in the Python Standard Library, (B) detect platform-specific APIs, and (C) assess the usage of platform-specific APIs.

A. Collecting APIs in the Python Standard Library. First, we inspected the *The Python Standard Library reference manual*[8] of the official language documentation, which summarizes all modules provided by the Python Standard Library in version 3.11. Next, we manually identified all modules that can be imported by client systems. In this process, we took special care to filter out built-in (written in C) modules, types, functions, and constants, which are not in the scope of this study. We detected 341 modules, and for each module, we mined their provided APIs. In this process, we identified 8,795 APIs, as summarized in Table 1. Most APIs are at the level of method (33.43%), function (24.84), and data (16.35%).

Table 1: APIs provided by the Python Standard Library.

API Level	Description	#	%
Method	Object method	2,940	33.43
Function	Module-level function	2,185	24.84
Data	Global data (variable or value)	1,438	16.35
Attribute	Object data attribute	1,073	12.20
Class	Class	942	10.71
Exception	Exception class	217	2.47
All		8,795	100.00

B. Detecting Platform-Specific APIs. APIs provided by the Python Standard Library may have *availability*[9] notes indicating their supported and unsupported platforms. This is the standard way to indicate the API availability, for example, *Availability: Windows* means that an API is specific to Windows. According to the Python documentation, *Availability: Unix* indicates that the API is generally supported by macOS. We collected this information and classified the APIs with availability notes as platform-specific. Among the 8,795 APIs provided by the Python Standard Library, we detected 1,841 platform-specific APIs.

C. Assessing the usage of platform-specific APIs. We designed and implemented an AST-based tool to parse source code and detect the usage of platform-specific APIs. Specifically, the tool detects the presence of the selected platform-specific APIs and their location in the source code, in test or production code (a Python source file with the substring "test" on its path is classified as *test*, otherwise it is classified as *production*). We run the proposed tool to detect platform-specific APIs in the 100 selected systems. In total, we detect over 19K usages of platform-specific APIs in all 100 projects.

2.3 Research Questions

2.3.1 RQ1: What is the availability of platform-specific APIs? First, we assess the availability of the platform-specific APIs provided by the Python Standard Library. We analyze the occurrence of platform-specific APIs by API level, module, and platform. **Rationale:** We

[5]https://github.com/ray-project/ray/blob/fc98a5f286877ce7f6241961aca0c9127bee21ad/python/ray/tune/experiment/trial.py#L169
[6]https://seart-ghs.si.usi.ch
[7]https://docs.python.org/3/library/os.html#os.listdrives
[8]https://docs.python.org/3.11/library/index.html
[9]https://docs.python.org/3.11/library/intro.html#notes-on-availability

aim to better understand to what extent platform-specific APIs happen in Python Standard Library and identify the key entities. So far, it is not clear the extension of those APIs.

2.3.2 RQ2: What is the usage of platform-specific APIs? Next, we explore the usage of platform-specific APIs by client systems. Here, we present the usage in three distinct views: by system, API, and module. We also present the data according to their location in the source code (test or production code). **Rationale:** We aim to better understand to what extent platform-specific APIs are consumed by real-world Python systems. If this is common, it may bring to light novel discussions, for example, whether developers are aware of the API restrictions and how to mitigate possible problems caused by the usage of platform-specific APIs.

3 RESULTS

3.1 RQ1: Availability of Platform-Specific APIs

Table 2 summarizes the detected platform-specific APIs. Overall, we find 1,841 (20.93%) platform-specific APIs in the 8,795 APIs provided by the Python Standard Library. The most common API levels are methods (715 APIs), function (363 APIs), and data (308 APIs). The highest proportion happens in exceptions: 80 out of 217 (36.87%).

Table 2: Platform-specific APIs by API level.

Pos	API Level	#APIs	Platform-Specific APIs	
			#	%
1	Method	2,940	715	24.32
2	Function	2,185	363	16.61
3	Data	1,438	308	21.42
4	Attribute	1,073	195	18.17
5	Class	942	180	19.11
6	Exception	217	80	36.87
All		8,795	1,841	20.93

Overall, we find platform-specific APIs in 51 out of 341 (15%) Python modules. Table 3 presents the modules with the most platform-specific APIs. The Python modules with the most platform-specific APIs are `asyncio` (300), `os` (220), and `ssl` (157).

Table 3: Platform-specific APIs by module.

Pos	Module	#	%
1	asyncio	300	16.30
2	os	220	11.95
3	ssl	157	8.53
4	socket	121	6.57
5	urllib.request	94	5.11
All		1,841	100.00

> ***Finding 1**: 21% of the APIs provided by the Python Standard Library are platform-specific. 15% of the modules contain at least one platform-specific API. The modules with the most platform-specific APIs are* `asyncio`*,* `os`*, and* `ssl`*.*

Table 4 details the availability of the platform-specific APIs. Notice the APIs are most unavailable on the platforms WASI (43.69%) and Emscripten (43.64%). We also find a significant number of APIs that are only available on certain operation systems: Unix (6.76%), Windows (2.12%), and Linux (1.94%). Overall, we find 17 different platforms with availability restrictions, as summarized in Table 5.

Table 4: Platform-specific APIs by platform.

Pos	Platform	#	%
1	not WASI	1,667	43.69
2	not Emscripten	1,665	43.64
3	Unix	258	6.76
4	Windows	81	2.12
5	Linux	74	1.94
-	Others	73	1.91
All		3,818	100.00

Table 5: Platforms with availability restrictions.

Linux, *Windows*, *macOS*, *Unix*, *AIX*, *Android*
Emscripten, *WASI*, *pthreads*, *POSIX*, *Solaris*, *VxWorks*
FreeBSD, *BSD*, *DragonFlyBSD*, *NetBSD*, *OpenBSD*

> ***Finding 2**: We find 17 different platforms with availability restrictions. The most frequent are WASI (43.69%), Emscripten (43.64%), Unix (6.76%), and Windows (2.12%).*

3.2 RQ2: Usage of Platform-Specific APIs

Next, we analyze the usage of the platform-specific APIs in the 100 selected projects. Overall, we find 19,288 usages of 683 platform-specific APIs in all 100 projects. On the median, each project uses 79.5 platform-specific APIs (the first quartile is 23.3 and the third quartile is 184.3). Table 6 details the top-5 projects with the most platform-specific APIs. The Ray project has the highest usage (2,267), followed by Salt (2,098) and AIOHTTP (1,173).

Table 6 also details *where* the platform-specific APIs are located in source code, that is, in production or test code. Interestingly, among the 19,288 usages of platform-specific APIs, we find that 10,152 (52.6%) are located in the production code, while 9,136 (47.4%) happen in the test code.

> ***Finding 3**: Platform-specific APIs are largely used in Python. We find 19,288 usages of 683 platform-specific APIs in all 100 projects, in both production (52.6%) and test code (47.4%).*

Table 6: Usage of platform-specific APIs.

Pos	Project	#	%	Test	Production
1	ray-project/ray	2,267	11.75	1,336	931
2	saltstack/salt	2,098	10.88	857	1,241
3	aio-libs/aiohttp	1,173	6.08	867	306
4	jina-ai/jina	1,162	6.02	889	273
5	ansible/ansible	888	4.60	308	580
All		19,288	100.00	9,136	10,152

Overall, the top-5 most used modules are `subprocess` (25.76%), `asyncio` (21.38%), `threading` (13.18%), `os` (10.40%), `socket` (10.26%). Table 7 summarizes the most used platform-specific APIs. The most used API is `asyncio.sleep`[10] (838 occurrences), which suspends the current task. Next, we have two APIs provided by the module `subprocess`. The API `subprocess.Popen`[11] executes a child program in a new process and has 798 occurrences. The API `subprocess.PIPE`[12] has 703 occurrences and it is a special value that indicates that a pipe should be opened. In Table 7, we also detail the frequency of the platform-specific APIs in test and production code. The usage of the platform-specific APIs is distinct among test and production code. For example, the API `asyncio.sleep` is mostly used in test code (83%), while the API `os.getenv`[13] is mostly used in production code (74%). The top-3 modules in the test code are: `asyncio`, `subprocess`, and `threading`, while in the production code are: `subprocess`, `asyncio`, and `os`.

> ***Finding 4****: The most used platform-specific APIs are* `asyncio.sleep`*,* `subprocess.Popen`*,* `subprocess.PIPE`*, and* `os.getenv`*. However, there is a difference in usage in test and production code:* `asyncio.sleep` *is mostly used in tests, while* `os.getenv` *is mostly used in production code.*

4 DISCUSSION AND IMPLICATIONS

Platform-specific APIs are widespread in the Python Standard Library but there is a lack of dedicated documentation. In RQ1, we found that 21% of the APIs provided by the Python Standard Library are platform-specific. Modules like `asyncio`, `os`, and `ssl` contain hundreds of platform-specific APIs. Moreover, we find 17 different platforms with availability restrictions, including mainstream OSs (Linux, Windows, macOS, and Unix), open-source Unix-like OSs (BSD, OpenBSD, FreeBSD, DragonFlyBSD, and NetBSD), proprietary OSs (Oracle Solaris, IBM AIX, and VxWorks), mobile OS (Android), WebAssembly platforms (Emscripten and WASI), and standards (POSIX and pthreads). Despite platform-specific APIs being widespread in the Python Standard Library, there is no documentation dedicated to their availability. The Python Standard Library documentation only provides a few notes regarding availability.[14] *Therefore, we recommend that dedicated documentation should be provided to better present the current state of availability and guide developers in charge of using platform-specific APIs. This kind of documentation could be auto-generated based on our dataset.*

Platform-specific APIs are largely used by Python systems but there is an absence of best practices and anti-patterns. In RQ2, we found over 19K usages of 683 platform-specific APIs in all 100 projects, in both production (52.6%) and test code (47.4%). We also detected that some APIs are more used in test code, while others are more adopted in production code. Given that platform-specific APIs are largely used by Python systems, developers would benefit from best practices to use them, both in test or production code. For example, the code presented in Figure 1 shows that developers may use multiple solutions (i.e., defensive coding) to call platform-specific APIs, however, it is not clear what are the possible solutions and most adopted ones. *Thus, our results provide the basis for the development of novel qualitative studies on how to properly use platform-specific APIs, revealing best practices and the anti-patterns that should be avoided.*

5 LIMITATION

This study focuses on the analysis of the platform-specific APIs provided by the Python Standard Library and their usage in popular projects hosted on GitHub. Therefore, our findings – as usual in empirical software engineering – may not be directly generalized to other projects or other programming languages. Further studies should be performed to better understand the platform-specific APIs in other software ecosystems.

6 RELATED WORK

APIs provide several benefits to users, such as feature reuse, productivity improvement, and reduction of development costs [9, 16, 17, 19]. API is a research topic broadly studied by prior literature, including API migration [2, 15], API deprecation [12, 18, 20–22, 25], and API evolution [5, 8, 10, 13, 14, 23, 24], to name a few. Some studies explore the compatibility issues caused by Android API evolution. In this case, researchers assess the challenges faced by Android developers to keep their applications working on multiple Android platforms [13, 14, 24]. To our knowledge, platform-specific APIs are not directly covered by the literature. Our study contributes to this research line by assessing the availability and usage of the platform-specific APIs provided by the Python Standard Library.

7 CONCLUSION AND FURTHER STEPS

In this paper, we provided an empirical study to assess the availability and usage of platform-specific APIs in Python. We analyzed the platform-specific APIs of the Python Standard Library and mined their usage in 100 popular systems. We found that 21% of the Python Standard Library APIs are platform-specific and 15% of the modules contain at least one. We also found 19,288 usages of platform-specific APIs that were detected across 100 projects, both in production (52.6%) and test code (47.4%). Lastly, we discussed practical implications for practitioners and researchers.

As future work, we plan to perform a qualitative analysis to better understand how platform-specific APIs are used in practice by developers and explore what are the best programming practices.

[10] https://docs.python.org/3/library/asyncio-task.html#asyncio.sleep
[11] https://docs.python.org/3/library/subprocess.html#subprocess.Popen
[12] https://docs.python.org/3/library/subprocess.html#subprocess.PIPE
[13] https://docs.python.org/3/library/os.html#os.getenv
[14] https://docs.python.org/3/library/intro.html#notes-on-availability

Table 7: Most used platform-specific APIs.

Pos	API	API Level	All		Test		Production	
			#	%	#	%	#	%
1	asyncio.sleep	function	838	4.35	698	83	140	17
2	subprocess.Popen	class	798	4.14	343	43	455	57
3	subprocess.PIPE	data	703	3.64	292	42	411	58
4	os.getenv	function	673	3.49	178	26	495	74
5	subprocess.check_output	function	615	3.19	318	52	297	48
All			19,288	100.00	9,136	-	10,152	-

We also plan to analyze the availability and usage of platform-specific APIs in other programming languages and libraries.

ACKNOWLEDGMENT

This research is supported by CAPES, CNPq, and FAPEMIG.

REFERENCES

[1] Anonymous Anonymous. November, 2023. *Platform-Specific APIs.* https://doi.org/10.5281/zenodo.10120107

[2] Lívia Barbosa and Andre Hora. 2022. How and why developers migrate Python tests. In *International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 538–548.

[3] Hudson Borges, Andre Hora, and Marco Tulio Valente. 2016. Understanding the Factors that Impact the Popularity of GitHub Repositories. In *International Conference on Software Maintenance and Evolution*. 334–344. https://doi.org/10.1109/ICSME.2016.31

[4] Hudson Borges and Marco Tulio Valente. 2018. What's in a GitHub Star? Understanding Repository Starring Practices in a Social Coding Platform. *Journal of Systems and Software* 146 (2018), 112–129. https://doi.org/10.1016/j.jss.2018.09.016

[5] Aline Brito, Marco Tulio Valente, Laerte Xavier, and Andre Hora. 2020. You Broke My Code: Understanding the Motivations for Breaking Changes in APIs. *Empirical Software Engineering* 25 (2020), 1458–1492.

[6] Ozren Dabic, Emad Aghajani, and Gabriele Bavota. 2021. Sampling Projects in GitHub for MSR Studies. In *International Conference on Mining Software Repositories (MSR)*. IEEE, 560–564. https://doi.org/10.1109/MSR52588.2021.00074

[7] GitHub-hosted runners. November, 2023. https://docs.github.com/en/actions/using-github-hosted-runners/about-github-hosted-runners/about-github-hosted-runners.

[8] Andre Hora, Romain Robbes, Marco Tulio Valente, Nicolas Anquetil, Anne Etien, and Stephane Ducasse. 2018. How do Developers React to API Evolution? A Large-Scale Empirical Study. *Software Quality Journal* 26, 1 (2018), 161–191.

[9] Dino Konstantopoulos, John Marien, Mike Pinkerton, and Eric Braude. 2009. Best principles in the design of shared software. In *International Computer Software and Applications Conference*. 287–292.

[10] Raula Gaikovina Kula, Daniel M German, Ali Ouni, Takashi Ishio, and Katsuro Inoue. 2018. Do developers update their library dependencies? An empirical study on the impact of security advisories on library migration. *Empirical Software Engineering* 23 (2018), 384–417.

[11] Maxime Lamothe, Yann-Gaël Guéhéneuc, and Weiyi Shang. 2021. A Systematic Review of API Evolution Literature. *ACM Computing Surveys (CSUR)* 54, 8 (2021), 1–36. https://doi.org/10.1145/3470133

[12] Li Li, Jun Gao, Tegawendé F Bissyandé, Lei Ma, Xin Xia, and Jacques Klein. 2018. Characterising deprecated android apis. In *International Conference on Mining Software Repositories*. 254–264.

[13] Mario Linares-Vásquez, Gabriele Bavota, Carlos Bernal-Cárdenas, Massimiliano Di Penta, Rocco Oliveto, and Denys Poshyvanyk. 2013. Api change and fault proneness: A threat to the success of android apps. In *Joint Meeting on Foundations of Software Engineering*. 477–487.

[14] Tarek Mahmud, Meiru Che, and Guowei Yang. 2022. Android API field evolution and its induced compatibility issues. In *International Symposium on Empirical Software Engineering and Measurement*. 34–44.

[15] Matias Martinez and Bruno Gois Mateus. 2020. How and Why did developers migrate Android Applications from Java to Kotlin? A study based on code analysis and interviews with developers. *arXiv preprint arXiv:2003.12730* (2020).

[16] Gabriel Menezes, Bruno Cafeo, and Andre Hora. 2021. How Are Framework Code Samples Maintained and Used by Developers? The Case of Android and Spring Boot. *Journal of Systems and Software* 1 (2021), 1–30.

[17] Simon Moser and Oscar Nierstrasz. 1996. The effect of object-oriented frameworks on developer productivity. *Computer* 29, 9 (1996).

[18] Romulo Nascimento, Eduardo Figueiredo, and Andre Hora. 2021. JavaScript API Deprecation Landscape: A Survey and Mining Study. *IEEE Software* 39, 3 (2021), 96–105.

[19] Steven Raemaekers, Arie van Deursen, and Joost Visser. 2012. Measuring software library stability through historical version analysis. In *IEEE International Conference on Software Maintenance (ICSM)*. 378–387. https://doi.org/10.1109/ICSM.2012.6405296

[20] Romain Robbes, Mircea Lungu, and David Röthlisberger. 2012. How do developers react to API deprecation? The case of a Smalltalk ecosystem. In *International Symposium on the Foundations of Software Engineering*. 1–11.

[21] Anand Ashok Sawant, Romain Robbes, and Alberto Bacchelli. 2018. On the reaction to deprecation of clients of 4 + 1 popular Java APIs and the JDK. *Empirical Software Engineering* 23, 4 (Aug. 2018), 2158–2197. https://doi.org/10.1007/s10664-017-9554-9

[22] Anand Ashok Sawant, Romain Robbes, and Alberto Bacchelli. 2019. To react, or not to react: Patterns of reaction to API deprecation. *Empirical Software Engineering* 24, 6 (Dec. 2019), 3824–3870. https://doi.org/10.1007/s10664-019-09713-w

[23] Laerte Xavier, Aline Brito, Andre Hora, and Marco Tulio Valente. 2017. Historical and impact analysis of API breaking changes: A large-scale study. In *International Conference on Software Analysis, Evolution and Reengineering*. IEEE, 138–147.

[24] Hao Xia, Yuan Zhang, Yingtian Zhou, Xiaoting Chen, Yang Wang, Xiangyu Zhang, Shuaishuai Cui, Geng Hong, Xiaohan Zhang, Min Yang, et al. 2020. How Android developers handle evolution-induced API compatibility issues: a large-scale study. In *International Conference on Software Engineering*. 886–898.

[25] Jing Zhou and Robert J Walker. 2016. API deprecation: a retrospective analysis and detection method for code examples on the web. In *International Symposium on Foundations of Software Engineering*. 266–277.