Challenges of SonarQube Plug-In Maintenance

Bence Barta, Günter Manz, István Siket
University of Szeged, Department of Software Engineering
Árpád tér 2. H-6720 Szeged, Hungary
{bartab,magun,siket}@inf.u-szeged.hu

Rudolf Ferenc
FrontEndART Software Ltd.
Zászló u. 3 I./5. H-6722 Szeged, Hungary
ferenc@frontendart.com

***Abstract*—The SONARQUBE™ platform is a widely used open-source tool for continuous code quality management. It provides an API to extend the platform with plug-ins to upload additional data or to enrich its functionalities. The *SourceMeter plug-in for SONARQUBE™ platform* integrates the SourceMeter static source code analyzer tool into the SONARQUBE™ platform, i.e., uploads the analysis results and extends the GUI to be able to present the new results. The first version of the plug-in was released in 2015 and was compatible with the corresponding SONARQUBE™ version. However, the platform – and what is more important, its API – have evolved a lot since then, therefore the plug-in had to be adapted to the new API. It was not just a slight adjustment, though, because we had to redesign and reimplement the whole UI and, at the same time, perform significant alterations in other parts of the plug-in as well. Besides, we examined the effect of the API evolution on other open-source plug-ins and found that most of them still remain compatible with the latest version, even if they have not been updated alongside the underlying API modifications. The reason for this is that these plug-ins use only a small part of the API that have not changed over time.**

***Index Terms*—SourceMeter, SonarQube, plug-in, API, evolution, software quality, metrics, clone detection, coding issues**

I. Introduction

In software development, it is very common to use third-party libraries and components during development. This creates dependencies between third-party component developers and users, so developers of such components should be very careful about changing the API. On the other hand, it is necessary to keep pace with the evolution of programming languages and technologies, and meet the needs that arise from the users. Besides, the component itself also evolves continuously, and we have not even considered any bug fixes or security risks. These changes are not always easy to follow and previous research has shown that they are not always justified either. Ko et al. [4] examined 8 Java library API documentations and evaluated the results against 2126 applications. They found that, on average, 3.6 APIs per month become obsolete, while the same amount is removed. Unfortunately, 39% of the cases do not specify which new APIs can replace the old functionality and it is rarely explained why the API became obsolete. Additionally, the outdated API was updated in 62% of the cases when the API was properly documented and in only 49% of the cases when the appropriate documentation was missing. Brito et al. [1] asked developers to justify all their API changes that break compatibility. They found that the reason for most of the compatibility-breaking modifications were generally inspired by the need to implement new features and evolutionary purposes. In addition, the two other common reasons were that they wanted to simplify the API and improve maintenance. However, there were cases where we cannot say what the reason of the change was. Zhou and Walker [10] found that an API item is much more frequently removed than marked as obsolete. This is typical for small projects where it would require too much extra work to maintain them. These API changes can be grouped into three categories: (1) increased resources for maintenance, (2) low impact or minor changes, and (3) other motivations.

The SONARQUBE™ platform [6] ("platform" in the following) is an open-source software quality assurance framework that provides an API to develop external plug-ins that allow uploading additional data and the creation of custom graphical user interfaces (GUI). The open-source SourceMeter plug-in [3], [9] ("SM plug-in" in the following) adds the results of the SourceMeter static command line source code analyzer [8] to the platform and extends it with custom GUI pages. The SM plug-in has not changed since its latest release (which was two years ago) but the platform has been evolving and newer versions were released. During this time, most of the APIs used by the SM plug-in have become obsolete, which means the SM plug-in cannot be used with the newer versions of the platform because backward compatibility was broken. This motivated us to update the earlier version of the SM plug-in to be compatible with the latest version of the platform. During development, we found that due to a significant change in the API, updating the SM plug-in requires significant changes and many of the previously developed solutions could no longer be reused. For example, the GUI component had to be retired and a completely new one had to be implemented

This prompted us to also examine 23 other open-source plug-ins to see how the API changes affected them and how much effort was required to maintain them.

II. SONARQUBE™ Platform Plug-In Development

SONARQUBE™ platform [2], [6] is an open-source quality assurance tool released by SonarSource [7] in 2009 and actively developed ever since. It helps users keep track of their systems and ensure quality. It supports more than 20

The research described has been carried out as part of the CROSSMINER Project, which has received funding from the European Union's Horizon 2020 Research and Innovation Programme under grant agreement No. 732223. Ministry of Human Capacities, Hungary grant 20391-3/2018/FEKUSTRAT is acknowledged.

languages and defines various source code metrics, coding rule violations, code duplications, and test coverages to qualify the system, and also provides an estimate of the technical debt.

The platform has a public API that allows the development of custom plug-ins. These plug-ins can be used to add new features to the system, expand the user interface, or upload additional data. However, we can only make such changes or additions if there is a suitable API for the purpose. These expansion options can be divided into three major groups:

- *Scanner*, which is responsible for running source code analyzers.
- *Process Unit*, which aggregates the results of the analysis (counts second level measurements, aggregates measurements[1], gives developers new issues, or saves the data).
- *Web Application*, which provides the user interface.

The platform has a Java API, only the GUI API is JavaScript. We can create a new plug-in by deriving our classes from classes (or interfaces) defined by the API. It is important to note that a class defined by a plug-in is only used by the platform in specified phases, e.g., `Sensor` classes are instantiated during code analysis and are only used in this phase. The supported API extensions include the following:

- A new programming language can be introduced as the target of an analysis.
- New coding rule violations can be defined.
- New file level metrics can be calculated.
- It is possible to design new pages to the GUI.

These extensions ensure that external plug-ins can be added to the platform that use the results of third-party tools. To do so, plug-ins have to execute the external tool, process the result, upload it to the platform, and new pages should be provided to display the results if needed.

A. The API of the Platform GUI

Documentation and examples describing the GUI are scarce. There is only a minimal description on the official site, with no related material located in the forums. The web API available in the GUI is documented well[2], but the JavaScript methods to be used with AJAX requests for accessing it are not so much[3]. Perhaps this is the reason why there are only a few plug-ins that extend the GUI. In addition to our plug-in, we have found only one other that adds a custom page to the platform [5]. That is why we briefly review how the GUI can be extended and what difficulties we had to deal with.

The first step in developing a GUI plug-in is to register a `Page` in the plug-in, resulting in a new "More" menu on the GUI. By clicking on this drop-down menu the titles of the added pages are listed. This part is done using Java code. Afterwards a JavaScript file must be specified, where a function ("specified function") is registered that manages the initialization of the new page.

[1]If we are analyzing a project written in multiple programming languages and each source code analyzer can only produce metrics for its language and these metrics must be summarized at higher level.

[2]https://docs.sonarqube.org/display/DEV/Web+API

[3]https://docs.sonarqube.org/display/DEV/Making+Ajax+Request

The biggest challenge we faced during the extension of the web application was that the platform only allows the specification of a single JavaScript function per newly added page, which has to manage the entire content, look, and behavior of that page. All HTML content has to be generated, and – if necessary – dynamically managed with event-listeners, created all inside that single function. Custom CSS styles may not be specified anywhere. It is also not possible to declare helper functions, since the scope in the file differs from where the platform executes the specified function. Consequently, all helpers would have to be declared in the global context in order to be callable, but declaring everything globally is considered a bad practice[4], and it might lead to accidental overrides of other functions already declared and used by the platform.

Lastly, it is also not possible to logically group independent pieces of code into separate JavaScript files, the platform API expects us to put all code into that one function in that one file. We developed a custom bootstrapping mechanism inspired by Jake Archibald[5], which uses dynamic JavaScript loading to inject new JavaScript or CSS source files into the HTML document and executes them in the specified order, effectively guaranteeing a similar environment as in any HTML document where scripts and CSS files can be included freely.

B. Evolution of the Platform

The platform evolves continuously and the associated API is evolving as well. Our aim was to resume development and improve our open-source plug-in which was compatible with platform version 4.5, and to make it work with version 6.7 [3], [9]. More than two years have passed between the releases of the two versions, and the platform has undergone spectacular changes. On the opening page of the older version, we only see some basic characteristics (such as the number of classes, functions, bugs, and the amount of cloned code) which makes it impossible to draw a more serious conclusion on the quality of the system. On the other hand, the opening page of the newer version displays the classification of the systems according to different aspects, and we will also get an "aggregated" evaluation of whether a given project passed the quality assurance check (*Passed*) or not (*Failed*).

At the same time, the platform's widget system was also eliminated, which allowed the user to compile unique dashboards in a straightforward way. Instead, we can implement custom pages in the new versions by programming their behavior in JavaScript. Another important change is that while the GUI was previously developed using the Ruby on Rails (RoR) framework, the new GUI is already developed in JavaScript.

Changes in the background of the API are less visible to users, but can cause headache for developers. For example, `Decorator` classes were used to modify the results calculated by the platform. They are completely removed from the system, so once the values are calculated they cannot be modified. `Initializer` classes were executed before

[4]https://www.w3.org/wiki/JavaScript_best_practices#Avoid_globals

[5]https://www.html5rocks.com/en/tutorials/speed/script-loading/#toc-dom-rescue

`Sensor` classes. The file system was also available in API version 4.5, but it is no longer in the newer versions. Our implementation used the file system at this stage, consequently we had to move all the functions into `Sensor` classes.

Beside the numerous API changes, there are some parts of the system that work the same way as before. In the settings page of the platform, the user has the opportunity to customize the code analysis. This includes changing the default threshold values for metrics, adding new parameters to the external plug-ins, and many more. These customization feature related API entries fortunately remained the same.

III. SourceMeter Plug-In

SourceMeter is a command line source code analyzer tool that supports C/C++, Java, C#, Python, and RPG languages [8]. It is able to calculate nearly 100 source code metrics, detect coding rule violations in the source code, and find Type-2 code clones (duplications). The results are presented in textual form (`.txt` and `.csv`), but a structured binary file containing all the results (accessible through an API) is also available for further processing.

The SM plug-in uploads the results of the SourceMeter tool to the platform, providing a new look for SourceMeter. We can configure the SourceMeter-related parameters on the GUI of the platform before the SM plug-in runs the SourceMeter command line tool as an external analyzer and uploads the results. As already described, the earlier version of the SM plug-in was designed for platform version 4.5 and has not been developed for two years. Meanwhile, new versions of the platform were released and its API has changed a lot, therefore the SM plug-in became obsolete and did not work with release 6.7. When we decided to support the new version, we encountered various difficulties because of the changes and the new API restrictions.

API changes affecting functionality: In the following paragraphs, we summarize the most important and most interesting API changes we have had to face and present a short description of our solutions.

Uploading "unsupported" results: The platform supports file level metrics only, while SourceMeter counts class and method level metrics as well. In the earlier version we introduced two new "special file kinds" in the platform, which represented the class and method level elements. This way we could upload classes, methods and their metrics, and at the same time, their child-parent relationships were also represented in the file system hierarchy. Luckily, although they became part of this hierarchy, they were "special files" under other files, therefore the platform did not display them on the GUI. On the other hand, the SM plug-in could recognize them, therefore it was able to present them (with their metrics) in an appropriate GUI placeholder. Unfortunately, the new API does not allow us to upload additional files, thus we cannot apply this trick now. To overcome this problem, we applied another workaround by introducing new system level metrics for the project. Since a metric value can store any kind of data, the SM plug-in extracts all similarly handled elements at a given level (e.g., classes, interfaces, enums, structs, and unions are handled as class level elements) and creates only a single JSON object that stores the elements and their metric values, then uploads it as one metric value. This way only four new metric values are uploaded (package, class, method and clone) for a programming language but a separate metric is defined for each programing language SourceMeter supports.

Code duplications: Both the platform and the SourceMeter toolchain can detect code duplications. In the earlier version, we were able to replace the built-in data, namely, the API allowed us to delete the results of the platform and to upload the code duplications found by SourceMeter. This is no longer possible, not even extending the platforms duplications with additional code duplications found by any external tools.

Unmodifiable platform results: Because of the lack of the above mentioned `Decorator` classes, the uploaded results cannot be modified which may lead to different kinds of anomalies. Maybe the biggest problem is that the code duplications detected by the platform cannot be replaced or extended. This means that the built-in duplication viewer shows only those duplications that were detected by the built-in analyzer of platform even if the external plug-in would provide better results. To solve this problem, we stored all code duplication results in a JSON object and designed and implemented a custom clone view GUI (see Figure 2). Another problem is that there are some metrics that are calculated both by the platform and SourceMeter, but since the SM plug-in is not able to replace these metrics, users might find different values for the same metric on different GUI pages.

GUI changes: The SM plug-in extends the platform with new GUI elements, including menus on the setup page as well as full custom pages. In the earlier version, we provided a dashboard for each project in order to view the metrics calculated by SourceMeter. In addition, for each class and method the metrics also appeared in the source code view and it was possible to navigate among them [3]. These features cannot be implemented in the newer versions. Instead, metrics now appear on a newly added page which replaces the old dashboard. The page is divided into 4 boxes (see Figure 1, only two boxes are presented), the first three of which show the metrics of packages, classes, and methods, and provide sorting per column, filtering for entries and/or metrics, and paging functionalities. To aid users in understanding what the values in the tables indicate, descriptions for each metric can be swiftly accessed by simply hovering over the name of any metric in the table header. Clicking the name of an item opens a new popup window displaying the source code belonging to that item, where the first line of the item is highlighted.

The SM plug-in can also deal with a project written in several languages and display its results on the dashboard. In this case, the set of items is merged together within each level, with an icon in front of their name indicating which language an individual item belongs to (see Figure 1).

The fourth box is more specific for clone classes and instances. In this box, clone instances are shown grouped

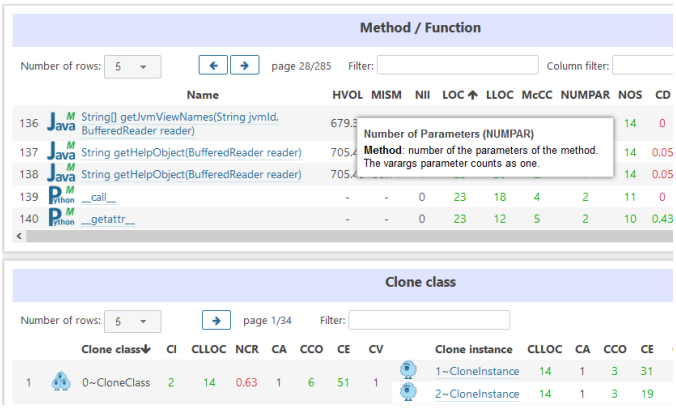

Fig. 1. SourceMeter Dashboard

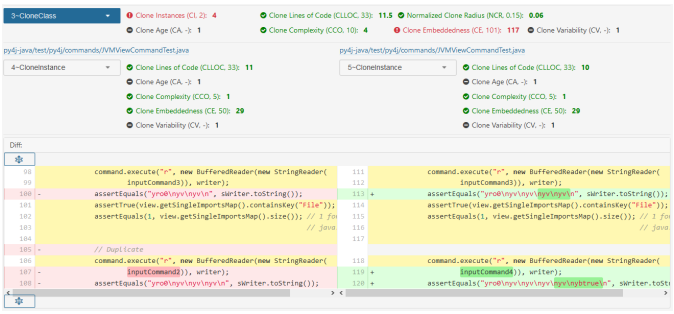

Fig. 2. SourceMeter Clone View

together next to the clone class they belong to[6]. The basic display of clones implemented in the platform needs improvement: it indicates duplicated lines in the code browser by placing gray stripes next to duplicated lines and showing and linking to the matching instances when hovering over these stripes. Using only this simple "clone browser", it is hard to get an overview of, e.g., how many clone classes and corresponding clone instances exist inside the project. The metrics calculated by the SM plug-in for each clone class and instance individually can help by providing this overview.

Additionally, a custom *Clone Viewer* has been released (see Figure 2). This new viewer puts emphasis on comparing instances and highlighting the differences between two clone instances using diffing software.

The SM plug-in also adds a new globally accessible page to the GUI containing the User's Guides for the SM plug-in and the underlying SourceMeter analyzer.

Unchanged API: Although there were lots of changes in the API, several functionalities remained the same. From the SM plug-in's point of view, it is convenient that the coding rule violations found by SourceMeter can be uploaded and the platform can present them as code issues. Besides, the setting pages have remained the same therefore we could reuse the existing implementation.

IV. Effect of the Platform API Evolution

The platform versioning strategy[7] has multiple goals. On one hand, they release often and early in order to get quick feedback from the SonarQube community. At the same time, they also release stable versions of the platform for companies whose main priority is to set up a very stable environment even if the price for such stable environments is missing out on the latest features. A central rule of their versioning strategy is that each two months a new version of the platform is released, which should increment the minor digit of the previous version (e.g. from 4.2 to 4.3). After three (or more) releases, a bug-fix version is released, and becomes the new long-term support. The major digit of the subsequent version is incremented to start a new cycle (e.g. from 5.6 to 6.0).

API deprecation strategy: The goal of the deprecation strategy is to make sure that deprecated APIs will be dropped without side-effects. The most important rules are:

- An API must be deprecated before being dropped.
- A deprecated API must be fully supported until its drop. For instance, the implementation of a deprecated method cannot be replaced by a `throw new UnsupportedOperationException()`.
- If an API is deprecated in version X.Y, this API will be dropped in version (X+2).0. For example, an API deprecated in 4.1 is supported in 4.2, 4.3, 5.0, 5.1, 5.2, 5.3 and is dropped in version 6.0.
- According to the versioning strategy, an API can remain deprecated before being dropped for 6 to 12 months.

There are many open-source community plug-ins for the platform. 66 plug-ins can be found on GitHub under SonarSource's GitHub organization page[8] but, for example, the community C++ plug-in can be found on a separate GitHub page called SonarOpenCommunity[9]. As we have seen, our plug-in upgrade was a big challenge, so we wanted to examine how the API changes of the platform affected the other plug-ins and how they dealt with it. It is difficult to answer this in general, therefore we examined the following questions:

- Does the platform version 6.7 start without any errors using the latest version of the plug-in?
- How active is the plug-in development community?
- How active is the development of these plug-ins?
- When did the last release happen?

Since there are lots of Community plug-ins, we chronologically ordered them and selected only the 22 most recently updated ones – with our SM plug-in as the 23rd.

When calculating the activity of development, we searched for the last commits in each repository in which the source code was modified in some significant way. This means that we did not include changes to the build process, nor the version updates of external libraries. If the source code has changed, but no changes have been made to the logical structure of the program (for example, exchanging licenses at the beginning

[6]Clone instances are the occurrences of code copies while clone classes group the corresponding clone instances.

[7]The description of the version strategy, rules, and deprecation is copied from the platform Versioning and API Deprecation homepage: https://docs.sonarqube.org/display/DEV/Versioning+and+API+Deprecation.

[8]https://github.com/SonarQubeCommunity

[9]https://github.com/SonarOpenCommunity

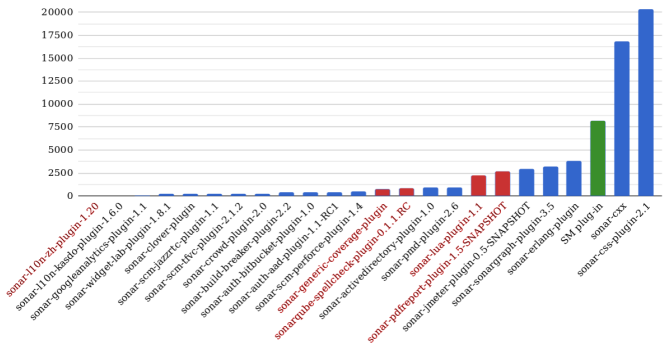

Fig. 3. The LOC of the selected plug-ins (SM plug-in is presented in green)

of the source files or rewriting the license's validity date), we did not consider it as a substantive change as they do not cause any bug fixes or feature additions to the plug-in and presumably modifications to the platform API would not have any influence on these types of changes.

Figure 3 shows the selected plug-ins ordered in increasing lines of code (LOC) values and the red names indicate that the plug-in does not work with version 6.7. Almost two-third of the plug-ins have less than 1000 lines, although there are several medium-sized and two extremely large plug-ins.

As shown in Figure 4, most of the plug-ins were successfully launched with version 6.7; only 5 of the 23 tested plug-ins were incompatible, which is only 22%. It is also interesting that most plug-ins have not been modified since 2016 – i.e., for 2 years – and they still work without any issues. We have found that the bulk of the plug-ins are small and simple, which is presumably why they are not affected by the changes made to the API, so there is no need to update them. This would explain how it is possible that the 2 or 3 years old code is still working. Additionally, none of them use the GUI API but simply upload data, mainly coding rule violations, so they are not affected by switching from RoR to JavaScript either. It is conspicuous that all but one of the plug-ins smaller than 1000 LOC still operate error-free now.

On the other hand, a few medium-sized plug-ins could not be started. For example, the SonarQube Lua Plugin (3,957 LOC) is one of the most well-maintained plug-ins in the list but the plug-in is incompatible with the current version for at least a year now, but it still has not been repaired so far. This suggests that the task is not simple, otherwise it would

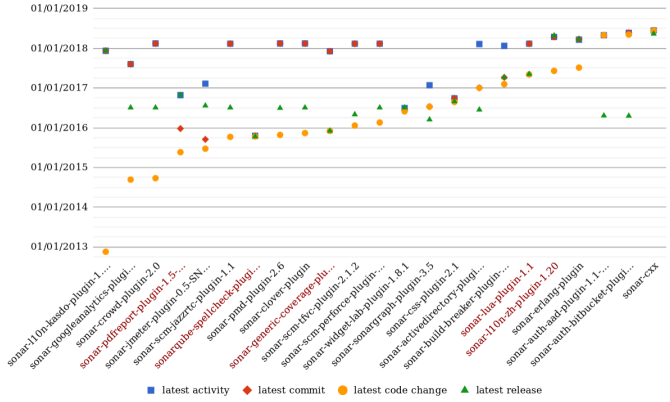

Fig. 4. The examined events associated with the plug-ins

probably have been solved by now. One of the largest and most actively developed plug-in is the open community C++ plug-in which does work with the latest version.

Based on this investigation, we have come to the conclusion that some APIs do not change, so many plug-ins work well even under low maintenance. However, 22% of the tested plug-ins are obsolete, including medium-sized and well-maintained plug-ins as well, suggesting that the API's change is significant, which is also supported by our own experience with the SM plug-in. Based on these, we can say that the change in the API of the platform is not negligible and requires continuous maintenance and development from plug-ins.

V. Conclusion

Originally, the SONARQUBE™ platform provided many possibilities for external plug-ins to extend existing functionality or upload additional data. However, many opportunities have been eliminated or transformed in recent years. Since the SourceMeter plug-in used major parts of the affected APIs (uploading or replacing data and files, new GUI elements), these changes caused big problems and we encountered serious difficulties when trying to keep the main functionalities of the plug-in during the upgrade. In this paper, we presented the most important API changes of the platform and explained the solutions and tricks we used to further develop the SM plug-in. We could see that some of the changes were only technical, but there were also a number of constraints in the changes that limit the possibilities of plug-ins. The examination of other plug-ins has shown that although the changes of the API do not cause problems for "simple" plug-ins, larger plug-ins require more effort to maintain compatibility.

The SourceMeter plug-in's source code, releases, and online demo and video is available at https://github.com/FrontEndART/SonarQube-plug-in.

References

[1] Aline Brito, Laerte Xavier, André C. Hora, and Marco Tulio Valente. Why and How Java Developers Break APIs. *CoRR, (TODO: Accepted at International Conference on Software Analysis, Evolution and Reengineering, SANER 2018)*, abs/1801.05198, 2018.

[2] G. Ann Campbell and Patroklos P. Papapetrou. *SonarQube in Action*. Manning Publications Co., Greenwich, CT, USA, 1st edition, 2013.

[3] Rudolf Ferenc, László Langó, István Siket, Tibor Gyimóthy, and Tibor Bakota. SourceMeter SonarQube plug-in. In *Proceedings of the 14th IEEE International Working Conference on Source Code Analysis and Manipulation (SCAM 2014)*, pages 77–82, Victoria, British Columbia, Canada, September 2014. IEEE Computer Society.

[4] Deokyoon Ko, Kyeongwook Ma, Sooyong Park, Suntae Kim, Dongsun Kim, and Yves Le Traon. API Document Quality for Resolving Deprecated APIs. In *21st Asia-Pacific Software Engineering Conf.*, pages 27–30, Jeju, South Korea, December 2014. IEEE Computer Society.

[5] Robert Willems of Brilman. Issue resolver Plugin for SonarQube. https://github.com/willemsrb/sonar-issueresolver-plugin.

[6] SonarQube Homepage. https://www.sonarqube.org/.

[7] SonarSource Homepage. https://www.sonarsource.com/.

[8] SourceMeter Homepage. https://www.sourcemeter.com/.

[9] SourceMeter plug-in for SONARQUBE platform. https://github.com/FrontEndART/SonarQube-plug-in.

[10] Jing Zhou and Robert J. Walker. API Deprecation: A Retrospective Analysis and Detection Method for Code Examples on the Web. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2016, pages 266–277, New York, NY, USA, 2016. ACM.