Dealing with SonarQube Cloud: Initial Results from a Mining Software Repository Study

Sabato Nocera
University of Salerno
Fisciano, Italy
snocera@unisa.it

Davide Fucci
Blekinge Institute of Technology
Karlskrona, Sweden
davide.fucci@bth.se

Giuseppe Scanniello
University of Salerno
Fisciano, Italy
gscanniello@unisa.it

***Abstract*—*Background:* Static Code Analysis (SCA) tools are widely adopted to enforce code quality standards. However, little is known about how open-source projects use and customize these tools.**
***Aims:* This paper investigates how GitHub projects use and customize a popular SCA tool, namely SonarQube Cloud.**
***Method:* We conducted a mining study of GitHub projects that are linked through GitHub Actions to SonarQube Cloud projects.**
***Results:* Among 321 GitHub projects using SonarQube Cloud, 81% of them are correctly connected to SonarQube Cloud projects, while others exhibit misconfigurations or restricted access. Among 265 accessible SonarQube Cloud projects, 75% use the organization's default *quality gate*, *i.e.,* a set of conditions that deployed source code must meet to pass automated checks. While 55% of the projects use the built-in quality gate provided by SonarQube Cloud, 45% of them customize their quality gate with different conditions. Overall, the most common quality conditions align with SonarQube Cloud's "Clean as You Code" principle and enforce security, maintainability, reliability, coverage, and a few duplicates on newly added or modified source code.**
***Conclusions:* Many projects rely on predefined configurations, yet a significant portion customize their configurations to meet specific quality goals. Building on our initial results, we envision a future research agenda linking quality gate configurations to actual software outcomes (*e.g.,* improvement of software security). This would enable evidence-based recommendations for configuring SCA tools like SonarQube Cloud in various contexts.**

***Index Terms*—Automation Policies, Coding Issues, Continuous Integration and Delivery, SonarCloud, SonarLint, SonarQube, Static Code Analysis tools.**

I. Introduction

Automation is a pillar of modern Software Engineering (SE) practices, such as DevOps and DevSecOps [1]. Particularly, the field of software security identifies a spectrum of automation approaches, from fixed policies (*i.e.,* explicitly embedded in a tool/system, without the possibility of changes), to customizable (*i.e.,* allowing custom configurations), to dynamic ones (*i.e.,* the most flexible that adapt at runtime) [2]. The impact of different automation policies has been extensively studied in the context of end-user security (*e.g.,* to suggest policies for the Windows operating system's automatic updates [3]). However, investigating automation policies—such as the ones embedded in Static Code Analysis (SCA) tool configurations—is also fundamental in SE for improving developers' experience [4].

SCA tools have become integral to modern software development, particularly in Continuous Integration/Continuous Delivery (CI/CD) pipelines, where they help enforce coding standards and detect quality issues (including security issues) early. Among these tools, SonarQube Cloud [5] stands out as a popular and widely used Software-as-a-Service (SaaS) platform that provides automated code reviews and quality assessments across multiple programming languages [6]–[11].

A central feature of the automation provided by SonarQube Cloud is the *quality gate* [12], which is a configurable set of conditions that code must satisfy to be considered acceptable. These conditions typically include thresholds for code coverage, code duplication, maintainability, reliability, and security ratings. SonarQube Cloud tool also offers a (predefined) built-in quality gate, known as the "Sonar way," which enforces best practices on added or modified source code, following the "Clean as You Code" principle [13]. Nonetheless, project administrators can also define (customized) non-built-in quality gates to better reflect the quality goals of their projects.

Despite the growing adoption of SonarQube Cloud in open-source development (*e.g.,* it is used by more than 22k GitHub users [14]), there is a limited empirical understanding of how developers configure and use it in practice [15], [16]. Questions remain on the extent to which predefined configurations are retained, how quality gates are customized, and which quality conditions are prioritized across projects. Answering these questions (*i.e.,* understanding how developers interact with SonarQube Cloud) is important, for example, to bridge the gap between tool design and practice, guide tool vendors in refining predefined configurations, and improve automation.

In this paper, we provide *initial results* from a mining study of 321 GitHub projects using SonarQube Cloud with the final goal of identifying challenges and opportunities in SE research and practice. Among other things, our results reveal a nuanced landscape: while many projects rely on predefined configurations (*e.g.,* organization's default quality gate or SonarQube Cloud's built-in quality gate), a significant portion use customized quality gates, reflecting diverse quality assurance strategies. These insights contribute to a better understanding of how SCA tools are operationalized. Our study lays the groundwork for evidence-based recommendations for configuring SCA tools and future research linking configuration choices to software outcomes (*e.g.,* improvement of software quality characteristics such as security).

979-8-3315-9147-2/25/$31.00 ©2025 IEEE

II. Background and Related Work

In this section, we provide background on SonarQube Cloud and related work investigating SCA tool configurations.

A. SonarQube Cloud

SonarQube Cloud (formerly known as SonarCloud) is a SaaS automated code review and SCA tool designed to detect coding issues in many programming languages [5]. It is conceived to be used in CI/CD pipelines, and it integrates with all leading CI/CD systems (including GitHub Actions). To enable SonarQube Cloud in the CI/CD pipeline of a software project, it is necessary to install and configure a component called *scanner* and connect that software project with a SonarQube Cloud project hosted on the SonarQube Cloud platform [17].

A GitHub project can connect to a SonarQube Cloud project by using one of the following GitHub Actions: `sonarcloud-github-action` [18] or `sonarqube-scan-action` [14].[1] To enable the connection, the GitHub project must specify the *keys* of the SonarQube Cloud project and its organization. These keys are specified via the `sonar.projectKey` and `sonar.organization` attributes, which can be provided either in a `sonar-project.properties` file or within the configuration of the GitHub Action. For instance, the GitHub project `https://github.com/apache/cloudberry` specified the SonarQube Cloud keys in its `sonar-project.properties` file: `sonar.projectKey=apache_cloudberry` and `sonar.organization=apache`.

Once SonarQube Cloud is configured into a CI/CD pipeline, upon the occurrence of specified events (*e.g.,* push or merge request), the scanner analyzes the application's source code against a set of conditions called *quality gate* [12]. These conditions may include thresholds for code coverage, code duplication, maintainability, reliability, and security ratings. Maintainability issues (or *code smells*) indicate patterns that make the source code confusing or difficult to maintain. Reliability issues (or *bugs*) are coding errors likely to cause failures at runtime. Security-related issues include both *vulnerabilities*, requiring immediate remediation, and *hotspots*, highlighting potentially risky code that should be reviewed. Each coding issue is assigned a severity level, with levels ranging from least to most severe: *info*, *minor*, *major*, *critical*, and *blocker*.

SonarQube Cloud shows a *Passed* status when the analysis meets or exceeds the quality gate conditions; otherwise, it shows a *Failed* status. Each SonarQube Cloud project has a single quality gate definition that is activated at any given time [19]. Whenever a SonarQube Cloud project is created, its quality gate is set to the *default*. Project administrators choose the default quality gate for their projects among those available. SonarQube Cloud provides for every project its own *built-in* quality gate, *i.e.,* "Sonar way." According to SonarQube Cloud documentation [5], the built-in quality gate is suitable for most projects; nonetheless, project administrators can also define and choose a *non-built-in* quality gate.

B. Empirical Studies on Static Code Analyzers configurations

A survey of software developers [15] (n = 1,263) using Static Application Security Testing tools shows that, despite their popularity, 54% do not configure them, using out-of-the-box rule sets. Among the developers who change predefined configurations, the vast majority enable or disable predefined rules rather than create new ones. These changes to predefined configurations are associated with prolonged experience with the tools. Interestingly, SonarQube was the most popular tool in the study, as reported by 59% of the participants.

Zampetti et al. [20] studied SCA and their configurations mined from CI/CD pipelines of 20 GitHub projects. From the projects' commit history, the authors extracted changes made to configuration files and extracted categories of rules activated or deactivated at least once. Their results show that many SCA tools are configured to break the build, while a small minority only raise warnings without interrupting the pipeline. In 17 projects, the configurations are customized by activating additional rules beyond the predefined ones; however, there is a variation, between 40 and 50%, of activated rules across tools. For five projects, the configurations are never changed after the first time, whereas for projects with continued changes over time, rules are never disabled or removed. From these results, developers' motivation to change configurations is to limit additional build-breaking checks that are not fully understood.

Besides affecting build pipelines, different SCA configurations can also impact the overall security of a software product. Piskachev et al. [21] conducted a user study involving 40 practitioners with varying roles, including developers and security experts, spanning from 3 to 10+ years of experience. The study compared the performance of participants in addressing known vulnerabilities in a software-under-test using the predefined SCA configuration vis-à-vis allowing them to customize the tool. The results show that participants who changed the configuration were able to address 76% of vulnerabilities—a 15% improvement over the predefined configuration. Furthermore, the authors collected qualitative evidence through post-task interviews to understand the strategies employed by participants to modify the tool configuration. The findings revealed that participants adopted an iterative approach—they activate a different subset of rules at each iteration and study their output to address possible vulnerabilities.

SCA tools configuration is reported as one of the main pain points, together with false negatives, in the results of a semi-structured interview with 20 practitioners from diverse backgrounds (security, product management, development) [22]. Conversely, participants expressed ease of configuration as one of the most important characteristics of an SCA tool.

Our contribution is transversal to the research introduced above. We provide initial evidence on how developers use a popular SCA tool (*i.e.,* SonarQube Cloud) in practice, and, based on it, we identify opportunities in SE research and practice to support developers when dealing with such tools.

[1] Although deprecated and planned for removal, `sonarcloud-github-action` is still used alongside its replacement, *i.e.,* `sonarqube-scan-action`, in GitHub projects.

III. Study Design

In this section, we present the design of our study.

A. Goal and Research Question

By leveraging the Goal-Question-Metric (GQM) method [23], we formalize the goal of our study as follows:

> **Analyze** the use of SonarQube Cloud **for the purpose of** *(i)* understanding usage patterns, *(ii)* assessing the degree of customization of quality gates, and *(iii)* identifying the conditions verified by such quality gates **from the point of view of** practitioners and researchers **in the context of** open-source projects publicly hosted on GitHub that use SonarQube Cloud as part of their CI/CD pipelines.

Based on the above-mentioned goal, we formulated the following three Research Questions (RQs):

RQ1. *What usage patterns characterize the use of SonarQube Cloud in the CI/CD pipelines of GitHub projects?*

This RQ aims to understand the usage patterns characterizing the use of SonarQube Cloud in the CI/CD pipelines of GitHub projects. We derived such patterns while exploring how GitHub projects connect to the projects hosted on SonarQube Cloud and incur eventual misconfigurations or errors.

RQ2. *To what extent SonarQube Cloud projects use default and/or built-in quality gates?*

This RQ aims to assess the extent to which SonarQube Cloud projects use the organization's default quality gate and SonarQube Cloud's built-in quality gate, as well as the proportion of projects that do not use them. In other words, this RQ investigates the degree of customization in the quality gates used by SonarQube Cloud projects.

RQ3. *What conditions are verified by the quality gates of SonarQube Cloud projects?*

This RQ aims to identify the quality conditions defined and verified by the quality gates of SonarQube Cloud projects. This allows for investigating which quality characteristics (*e.g.,* maintainability, reliability, or security) developers choose to prioritize in CI/CD pipelines.

B. Study Context and Planning

The context of our study consists of open-source software projects publicly hosted on GitHub and using SonarQube Cloud in their CI/CD pipelines to enforce code quality standards. We considered GitHub because it is currently the most popular version-control and software-development-hosting platform for personal and professional use [24].

To search for software projects using SonarQube Cloud, we leveraged GitHub's dependency graph feature [25]. It allows retrieving all the dependencies and dependents of a given software repository, *i.e.,* the repositories a given repository depends on and the repositories that depend on it, respectively. The information about the dependencies and dependents of a repository is automatically inferred from the manifest and lock files, *i.e.,* files specifying project dependencies such as `.yaml` for GitHub Actions. In our case, we leveraged GitHub's dependency graph to retrieve all GitHub repositories dependent on `sonarcloud-github-action` [18] or `sonarqube-scan-action` [14]. We retrieved 34,975 GitHub repositories dependent on `sonarcloud-github-action` and 12,986 dependent on `sonarqube-scan-action`.

The analysis of data retrieved from GitHub can lead to conclusions not representative of the open-source projects intended to be investigated [26]. To deal with such an issue, we selected those repositories that met the following criteria:

1) *Not archived and with at least one commit made in the month preceding the query date (16 April 2025).* This was to avoid selecting software projects that were inactive [26].
2) *With at least 100 stars, at least 100 commits, and at least one fork.* This was to mitigate the risk of selecting personal projects [26]. We opted for such thresholds as they are common in mining software repository studies (*e.g.,* [27]–[29]).
3) *Not fork.* This was to limit the risk of selecting duplicate projects [26]. Forks inherit most of their data from the parent repository [30] and are likely inactive [31].

We ended up with 231 repositories dependent on `sonarcloud-github-action` and 149 dependent on `sonarqube-scan-action`. Since a repository could depend on both `sonarcloud-github-action` and `sonarqube-scan-action`, we discarded 59 duplicates among the GitHub repositories retrieved with GitHub's dependency graph. Eventually, we analyzed 321 unique GitHub projects using SonarQube Cloud through `sonarcloud-github-action` and/or `sonarqube-scan-action`.

C. Data Analysis

This section presents the data analysis arranged by RQ.

RQ1. We performed the following steps for each project:

1) We examined the content of the `sonar-project.properties` file and GitHub Actions files to identify the keys required for connecting the GitHub project with the SonarQube Cloud project.
2) We manually inspected the content of the repository to observe any pattern in the usage of SonarQube Cloud. We searched SonarQube Cloud references through the GitHub Code Search feature available for every repository; we used "`sonar`" as a search string and examined the search results (*e.g.,* files in which that string was found). If we did not find search results this way, we also examined the output of GitHub Actions workflows to gain any possible insights on the usage of SonarQube Cloud. Among other things, this allowed us to check and, eventually, correct the SonarQube Cloud keys that had been previously retrieved. We collected information about usage patterns through *open-coding* [32]. The first author conducted it and iteratively refined the resulting codes.
3) We verified whether the SonarQube Cloud project connected to the GitHub project could be queried through the SonarQube Cloud API [33]. This was to understand

TABLE I: Usage patterns of SonarQube Cloud.

Usage Pattern	Count (%)
SonarQube Cloud connected correctly	260 (81%)
SonarQube Cloud project absent or private	47 (14.6%)
SonarQube Cloud keys retrieved from GitHub Actions workflows	26 (8.1%)
GitHub project connected to several SonarQube Cloud projects	17 (5.3%)
SonarQube Cloud keys defined through environment variables	17 (5.3%)
SonarQube Cloud keys not found	12 (3.7%)
Integration of SonarQube Cloud with SonarLint	11 (3.4%)
SonarQube Cloud keys retrieved through GitHub Code Search	8 (2.5%)
Integration of SonarQube Cloud with Gradle	7 (2.2%)
SonarQube Cloud keys defined incorrectly in files	6 (1.9%)
Integration of SonarQube Cloud with Maven	4 (1.2%)
SonarQube Server used in place of SonarQube Cloud	3 (0.9%)
SonarQube Cloud project not configured	1 (0.3%)
SonarQube Cloud keys commented out	1 (0.3%)

if the identified SonarQube Cloud project allowed users outside its organization to access information about its code standards, including quality gates.

RQ2. We exploited the SonarQube Cloud API [33] to collect data on the quality gates used by the identified and accessible SonarQube Cloud projects. For each project, we determined whether the associated quality gate was the default one and whether it was built-in. We then computed the number and percentage of projects using quality gates that were default or non-default, built-in or non-built-in. Specifically, we analyzed all combinations of these attributes, distinguishing between quality gates that were: *(i)* both default and built-in, *(ii)* default but not built-in, *(iii)* built-in but not default, and *(iv)* neither default nor built-in.

RQ3. We exploited the SonarQube Cloud API [33] to collect data on the conditions verified by the quality gates of the identified SonarQube Cloud projects. We categorized each condition based on the classification provided by SonarQube Cloud documentation (*e.g.,* complexity, coverage, security) [34]. For each condition and each category, we counted the number and percentage of projects whose quality gates included it.

IV. Results

Below, we report the results by RQ.

A. RQ1. What usage patterns characterize the use of SonarQube Cloud in the CI/CD pipelines of GitHub projects?

In Table I, we report the identified usage patterns of SonarQube Cloud by GitHub projects. In the first place, 81% of the GitHub projects were connected correctly to SonarQube Cloud. In the other projects, we acknowledged the following problems with the connection to SonarQube Cloud:

- The SonarQube Cloud project was absent or made private in 14.6% of the GitHub projects. SonarQube Cloud does not explain if a given project is inaccessible because it does not exist or the user is not granted access permissions.
- We could not find SonarQube Cloud keys in 3.7% of the GitHub projects. We found no reference to SonarQube Cloud through either GitHub Code Search or GitHub Actions.
- The SonarQube Cloud project was not correctly configured in 0.3% of the GitHub projects, and, in another 0.3% of the GitHub projects, the SonarQube Cloud keys were commented out in the GitHub Action file (meaning that the GitHub Action could not be executed anyway).

During the manual inspection of the repositories, we retrieved some SonarQube Cloud keys from sources other than the `sonar-project.properties` file or GitHub Actions files. We retrieved SonarQube Cloud keys from the output of GitHub Actions workflows in 8.1% of the GitHub projects and through GitHub Code Search in 2.5% of them. We also acknowledged that 5.3% of the GitHub repositories defined SonarQube Cloud keys through environment variables (*e.g.,* `${{env.sonar-project-key}}`). We also observed that, in 1.9% of GitHub repositories, SonarQube Cloud keys were defined incorrectly, meaning that they could not refer to an existing SonarQube Cloud project.

Some GitHub projects integrated SonarQube Cloud with other technologies. In detail, 2.2% of GitHub projects integrated SonarQube Cloud with the *Gradle* package manager and 1.2% with the *Maven* package manager; the former declared the SonarQube Cloud keys in the `pom.xml` files, while the latter declared them in the `build.gradle` files. Also, 3.4% of GitHub projects connected SonarQube Cloud with *SonarLint* [35]. The latter is a linter, also known as *SonarQube for IDE*, that can be plugged into IDEs and is developed by the same organization as SonarQube Cloud, namely SonarSource [36]. SonarLint can detect a subset of SonarQube Cloud issues while developers code, helping them address such issues before even committing the code. Connecting SonarLint with SonarQube Cloud allows (parts of) the configuration of the two tools to be shared. There is also another version of SonarQube Cloud, namely *SonarQube Server* [37], which is used *on-premises*, *i.e.,* it is installed and managed by an organization's own physical hardware. We identified the use of SonarQube Server in place of SonarQube Cloud in 0.9% of the GitHub projects.

We found that 5.3% of GitHub projects were connected and using more than one SonarQube Cloud project. For example, the GitHub project `https://github.com/WE-Kaito/digimon-tcg-simulator` was connected to two SonarQube Cloud projects: `https://sonarcloud.io/summary/overall?id=we-kaito_digimon-tcg-simulator-backend` and `https://sonarcloud.io/summary/overall?id=we-kaito_digimon-tcg-simulator-frontend`; the former seems to be related to the back-end of the project, while the latter to its front-end, suggesting that different components of the same software project may require different code standards.

As a result, we found that 321 GitHub projects were correctly connected to 328 SonarQube Cloud projects. This means that there were some GitHub projects connected to more than one SonarQube Cloud project. Among the SonarQube Cloud projects, 63 could not be queried through the SonarQube Cloud API due to a lack of permission (*i.e.,* we were not part of the organization owning those SonarQube Cloud projects). Those 63 SonarQube Cloud projects were connected to 30

TABLE II: SonarQube Cloud projects using and not using default and/or built-in quality gates.

Quality Gate		Default: True	Default: False	*Total*
Built-in:	**True**	141 (53%)	4 (2%)	*145 (55%)*
	False	57 (22%)	63 (24%)	*120 (45%)*
Total		*198 (75%)*	*67 (25%)*	*265 (100%)*

GitHub projects, owned by 21 different GitHub users. We attempted to find reasons why these SonarQube Cloud projects restricted access through the SonarQube Cloud API. We speculate that, in most cases, this might be because the project operates in a security-sensitive application domain (*e.g.,* e-commerce or cryptocurrency) and, therefore, a willingness to limit available information on such projects. For example, five of these GitHub projects were owned by *bitwarden* [38], an organization providing security solutions such as password and secret managers. Eventually, we could retrieve the information about the code standards, including quality gates, of 265 SonarQube Cloud projects, which were connected to a total of 230 GitHub projects.

B. RQ2. To what extent SonarQube Cloud projects use default and/or built-in quality gates?

We analyzed the use of default and/or built-in quality gates among the 265 open-source projects using SonarQube Cloud (see Table II). Our analysis reveals that 141 projects (53%) used a quality gate that was both default and built-in. Conversely, 63 projects (24%) applied a quality gate that was neither default nor built-in, reflecting a fully customized configuration for those projects. Other 57 projects (22%) used the default quality gate but with a non-built-in version, while only 4 projects (2%) applied the built-in quality gate without being set as the default. In total, 198 projects (75%) relied on the default quality gate, and 145 (55%) used the built-in one. These findings suggest that while most projects rely on default and built-in configurations—likely due to their convenience and general applicability—a non-negligible number of projects customize those configurations. In particular, 67 projects (25%) did not retain the default quality gate of their organization, and 120 projects (45%) followed a customized configuration for the quality gate.

C. RQ3. What conditions are verified by the quality gates of SonarQube Cloud projects?

We found a total of 33 different quality conditions being verified by the quality gates of SonarQube Cloud projects. We report these conditions in Table III, along with the number (and percentage) of projects adopting each, grouped by the category indicated by SonarQube Cloud [34]. We have to mention that a given condition could be verified differently across projects by applying different thresholds. For example, one SonarQube Cloud project may consider the condition *vulnerabilities* met only when the number of vulnerabilities is zero, whereas another may allow up to one vulnerability.

The most commonly adopted conditions are those included in the built-in quality gate (see bold entries in Table III) and are applied to *new code*. In fact, SonarQube Cloud enforces the *Clean as You Code* practice, which is based on the principle that new code (*i.e.,* code that was added or modified recently) needs to comply with quality standards [13]. Since adding new code usually involves changes in existing code, analyzing and cleaning new code enables the gradual improvement of the overall quality of the codebase.

TABLE III: Identified quality conditions (in bold those in the built-in quality gate).

Category	Condition	Count (%)
Complexity	Cognitive complexity	2 (0.75%)
Coverage	Condition coverage on new code	12 (4.53%)
	Coverage on new code	**205 (77.36%)**
	Coverage	25 (9.43%)
	Unit test success density	1 (0.38%)
	Uncovered lines on new code	1 (0.38%)
Duplications	**Duplicated lines density on new code**	**234 (88.30%)**
	Duplicated lines density	18 (6.79%)
	Duplicated block on new code	1 (0.38%)
	Duplicated lines on new code	1 (0.38%)
Issues	Issues on new code	2 (0.75%)
	Blocker issues on new code	8 (3.02%)
	Blocker issues	4 (1.51%)
	Critical issues on new code	4 (1.51%)
	Critical issues	6 (2.26%)
	Major issues on new code	2 (0.75%)
	Minor issues on new code	1 (0.38%)
Maintainability	**Maintainability rating on new code**	**236 (89.06%)**
	Maintainability rating	16 (6.04%)
	Code smells	1 (0.38%)
	Code smells on new code	8 (3.02%)
Reliability	**Reliability rating on new code**	**235 (88.68%)**
	Reliability rating	18 (6.79%)
	Bugs	7 (2.64%)
	Bugs on new code	10 (3.77%)
Security	**Security rating on new code**	**241 (90.94%)**
	Security rating	21 (7.92%)
	Vulnerabilities	7 (2.64%)
	Vulnerabilities on new code	5 (1.89%)
Security Review	Security review rating on new code	4 (1.51%)
	Security review rating	3 (1.13%)
	Security hotspots reviewed	14 (5.28%)
	New security hotspots reviewed	**232 (87.55%)**

Among the conditions of the built-in quality gate, *security rating on new code* is the most prevalent, enforced in 90.94% of projects. It assesses newly introduced vulnerabilities on a five-level scale from A (no vulnerabilities) to E (at least one blocker vulnerability). Next, *maintainability rating on new code* appears in 89.06% of quality gates. It assigns a grade from A to E by comparing the estimated effort for fixing the detected maintainability issues (technical debt) with the effort needed to develop the same code. *Reliability rating on new code*, included in 88.68% of quality gates, similarly grades newly introduced bugs from A (no bugs) to E (at least one blocker bug). In 88.30% of quality gates, *duplicated lines density on new code* was applied to verify the proportion of duplicate lines within the newly added or modified code. *New security hotspots reviewed*, enforced in 87.55% of projects, verifies the percentage of reviewed security hotspots on new code. Finally, we found *coverage on new code* in 77.36% of quality gates. It combines line and condition coverage to evaluate the extent to which new or updated code is covered.

The conditions included in the built-in quality gate are also widely adopted in non-built-in quality gates. Considering that 45% of SonarQube Cloud projects use non-built-in quality gates, the least adopted built-in condition (*coverage on new code*) was enforced by 77.36% of projects, while the most adopted one (*security rating on new code*) was enforced by 90.94% of projects. This difference may also reflect the prioritization of certain quality characteristics over others (*e.g.,* security over code coverage).

Beyond the built-in conditions, Table III shows various other quality conditions used across projects. However, these are adopted with much lower frequency, *i.e.,* each by less than 10% of projects. This suggests that, although some projects may use quality gates with custom conditions, they tend to rely primarily on those conditions enforced by SonarQube Cloud.

V. Discussion

This section discusses the results, highlighting their implications and presenting potential threats to validity.

A. Implications of the Results

One of the main results of our study indicates that most practitioners do not change the predefined configuration of SonarQube Cloud, which is coherent with past research [15], [20]. On the other hand, a significant portion of the analyzed projects customize SonarQube Cloud configurations. This seems to suggest that predefined SCA tool configurations might not always be suitable and projects might require *dynamic configurations* [2]. To better understand this phenomenon, researchers should investigate the needs and motivation for deviating from predefined configurations (*e.g.,* through qualitative studies). Similarly, approaches based on machine learning and artificial intelligence could be leveraged to create configurations based on such needs dynamically.

Our results also encourage researchers to investigate SCA tool customization patterns. For example, different contexts could drive the prioritization of different parts of a configuration (*e.g.,* enforcing quality conditions related to security rather than maintainability). Researchers could also be interested in developing taxonomies capturing different customization patterns. This would support the improvement of predefined SCA tool configurations, as well as their customization.

Despite existing research showing that integrating several SCA tools improves software outcomes (*e.g.,* vulnerability detection [39] or software bill of material generation [40]), we found little evidence of SonarQube Cloud integrations with other tools, such as SonarLint. We foster research to investigate the issues arising when integrating several SCA tools with respect to their configurations (*e.g.,* conflicting rules or conditions). Accordingly, a further avenue of research is devising dynamic configurations tailored for the combined use of multiple SCA tools, rather than a single SCA tool.

B. Threats to Validity

We acknowledge some threats to the validity of our results. In the following, we discuss these threats according to the guidelines by Wohlin *et al.* [41]. We would like to mention that no threats to internal validity are presented because we did not investigate causal relationships.

Construct Validity. Our identification of SonarQube Cloud usage relies on GitHub's dependency graph and manual inspection of configuration files. Similarly to past studies (*e.g.,* [42], [43]), we used GitHub's dependency graph to retrieve repositories, in our case dependent on `sonarcloud-github-action` and/or `sonarqube-scan-action`. GitHub's dependency graph is prone to inaccuracies that can affect the results [44], [45].

Conclusion Validity. Our analysis's correctness depends on the SonarCloud API's accuracy and the interpretation of quality gate configurations. We mitigated this possible threat by cross-validating data sources and manually inspecting results and repositories.

External Validity. We studied open-source GitHub projects meeting specific activity and popularity thresholds. As such, our findings may not be generalized to private or enterprise projects or to GitHub projects with different characteristics.

VI. Remarks and Vision for Future Research

This paper presents initial results from a mining software repository study on how open-source developers use and configure a popular Static Code Analysis (SCA) tool, namely SonarQube Cloud. By analyzing 321 GitHub projects, we found that, while most projects rely on predefined SonarQube Cloud configurations (*i.e.,* default and built-in quality gates), a significant portion of those projects customize their configurations to better align with specific quality goals. The most commonly enforced conditions—such as security, maintainability, and reliability ratings on new code—reflect a strong adherence to SonarQube's "Clean as You Code" principle.

Our findings highlight that, while predefined configurations offer convenience and broad applicability, many projects require customized SCA tool configurations to meet their unique needs. This suggests that many developers are actively shaping how SCA tools like SonarQube Cloud are used in their development workflows.

As a vision for future work, we propose investigating the effectiveness of different quality gate configurations by linking them to downstream software outcomes, such as the improvement of software quality characteristics like security. This investigation could provide actionable insights into which configurations yield the greatest benefits in practice, ultimately guiding tool developers and practitioners toward more evidence-based quality assurance strategies.

Data Availability

The data are available online [46].

Acknowledgment

This project has been financially supported by the European Union NEXTGenerationEU project and by the Italian Ministry of the University and Research MUR, a Research Projects of Significant National Interest (PRIN) 2022 PNRR, project n. D53D23017310001 entitled "Mining Software Repositories for enhanced Software Bills of Materials" (MSR4SBOM), and the SEcure software engineering through Sensible AutoMation (SESAM) founded by KKS.

References

[1] I. Ozkaya, “Are devops and automation our next silver bullet?” *IEEE Software*, vol. 36, no. 4, pp. 3–95, 2019.

[2] *Security automation considered harmful?*, vol. Proceedings of the 2007 Workshop on New Security Paradigms, 2008.

[3] *In Control with no Control: Perceptions and Reality of Windows 10 Home Edition Update Features*. Reston, VA: Internet Society, 2019.

[4] A. Noda, M.-A. Storey, N. Forsgren, and M. Greiler, “Devex: What actually drives productivity?” *Communications of the ACM*, vol. 66, no. 11, pp. 44–49, 2023.

[5] SonarSource. (2025) Documentation. [Online]. Available: https://docs.sonarsource.com/sonarqube-cloud/

[6] G. Digkas, M. Lungu, P. Avgeriou, A. Chatzigeorgiou, and A. Ampatzoglou, “How do developers fix issues and pay back technical debt in the apache ecosystem?” in *International Conference on software analysis, evolution and reengineering*. IEEE, 2018, pp. 153–163.

[7] S. Nocera, S. Romano, R. Francese, R. Burlon, and G. Scanniello, “Managing vulnerabilities in software projects: the case of ntt data,” in *Proceedings of Euromicro Conference on Software Engineering and Advanced Applications*, 2023, pp. 247–253. [Online]. Available: https://doi.org/10.1109/SEAA60479.2023.00046

[8] S. Nocera, S. Romano, R. Francese, and G. Scanniello, “Training for security: Results from using a static analysis tool in the development pipeline of web apps,” in *Proceedings of International Conference on Software Engineering: Software Engineering Education and Training*. ACM, 2024. [Online]. Available: https://doi.org/10.1145/3639474.3640073

[9] M. T. Baldassarre, V. Lenarduzzi, S. Romano, and N. Saarimäki, “On the diffuseness of technical debt items and accuracy of remediation time when using sonarqube,” *Inf. Softw. Technol.*, vol. 128, p. 106377, 2020.

[10] S. Nocera, S. Romano, R. Francese, and G. Scanniello, “Software engineering education: Results from a training intervention based on sonarcloud when developing web apps,” *Journal of Systems and Software*, vol. 222, p. 112308, 2025. [Online]. Available: https://doi.org/10.1016/j.jss.2024.112308

[11] S. Nocera, S. Romano, D. Di Nucci, R. Francese, F. Palomba, and G. Scanniello, “Do static analysis tools improve awareness and attitude toward secure software development?” in *International Conference on the Quality of Information and Communications Technology*. Springer, 2024, pp. 399–407. [Online]. Available: https://doi.org/10.1007/978-3-031-70245-7_28

[12] SonarSource. (2025) Quality gates. [Online]. Available: https://docs.sonarsource.com/sonarqube-cloud/improving/quality-gates/

[13] ——. (2025) Setting up Clean as You Code. [Online]. Available: https://docs.sonarsource.com/sonarqube-cloud/digging-deeper/clean-as-you-code/introduction/

[14] ——. (2025) SonarSource/sonarqube-scan-action. [Online]. Available: https://github.com/SonarSource/sonarqube-scan-action

[15] G. Bennett, T. Hall, S. Counsell, E. Winter, and T. Shippey, “Do developers use static application security testing (sast) tools straight out of the box? a large-scale empirical study,” in *International Symposium on Empirical Software Engineering and Measurement*, 2024.

[16] C. Vassallo, S. Panichella, F. Palomba, S. Proksch, H. C. Gall, and A. Zaidman, “How developers engage with static analysis tools in different contexts,” *Empir. Softw. Eng.*, vol. 25, pp. 1419–1457, 2020.

[17] SonarSource. (2025) Open Source Projects - SonarQube Cloud. [Online]. Available: https://sonarcloud.io/explore/projects/

[18] ——. (2025) SonarSource/sonarcloud-github-action: Deprecated. Use https://github.com/SonarSource/sonarqube-scan-action instead. [Online]. Available: https://github.com/SonarSource/sonarcloud-github-action

[19] ——. (2025) Setting your standards. [Online]. Available: https://docs.sonarsource.com/sonarqube-cloud/standards/overview/

[20] F. Zampetti, S. Scalabrino, R. Oliveto, G. Canfora, and M. Di Penta, “How open source projects use static code analysis tools in continuous integration pipelines,” vol. International Conference on Mining Software Repositories. IEEE, 2017, pp. 334–344.

[21] G. Piskachev, M. Becker, and E. Bodden, “Can the configuration of static analyses make resolving security vulnerabilities more effective? - a user study,” *Empir. Softw. Eng.*, vol. 28, no. 5, p. 2, 2023.

[22] A. S. Ami, K. Moran, D. Poshyvanyk, and A. Nadkarni, “” false negative-that one is going to kill you”: Understanding industry perspectives of static analysis based security testing,” in *2024 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2024, pp. 3979–3997.

[23] V. R. Basili, *Software modeling and measurement: the Goal/Question/Metric paradigm*. University of Maryland, 1992.

[24] Stack Overflow. (2022) Developer Survey. [Online]. Available: https://survey.stackoverflow.co/2022/

[25] GitHub. (2025) About the dependency graph - GitHub Docs. [Online]. Available: https://docs.github.com/en/code-security/supply-chain-security/understanding-your-software-supply-chain/about-the-dependency-graph

[26] E. Kalliamvakou, G. Gousios, K. Blincoe, L. Singer, D. German, and D. Damian, “The promises and perils of mining github,” in *Proceedings of Mining Software Repositories*. ACM, 2014, pp. 92–101.

[27] S. Nocera, S. Romano, R. Francese, and G. Scanniello, “A large-scale fine-grained empirical study on security concerns in open-source software,” in *Proceedings of Euromicro Conference on Software Engineering and Advanced Applications*, 2023, pp. 418–425. [Online]. Available: https://doi.org/10.1109/SEAA60479.2023.00069

[28] M. H. Ibrahim, M. Sayagh, and A. E. Hassan, “A study of how docker compose is used to compose multi-component systems,” *Empir. Softw. Eng.*, vol. 26, pp. 1–27, 2021.

[29] M. Alfadel, D. E. Costa, and E. Shihab, “Empirical analysis of security vulnerabilities in python packages,” *Empir. Softw. Eng.*, vol. 28, no. 3, p. 59, 2023.

[30] M. Hadian, S. Brisson, B. Adams, S. Ghari, E. Noei, M. Fokaefs, K. A. Lyons, and S. Zhou, “Exploring trends and practices of forks in open-source software repositories.” in *CASCON*, 2022, pp. 120–129.

[31] S. Stanciulescu, S. Schulze, and A. Wasowski, “Forked and integrated variants in an open-source firmware project,” in *2015 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2015, pp. 151–160.

[32] M. B. Miles and A. M. Huberman, *Qualitative data analysis: An expanded sourcebook*. sage, 1994.

[33] SonarSource. (2025) Web API - SonarQube Cloud. [Online]. Available: https://sonarcloud.io/web_api/api/

[34] ——. (2025) Measures and metrics. [Online]. Available: https://docs.sonarsource.com/sonarqube-cloud/digging-deeper/metric-definitions/

[35] ——. (2025) Linter IDE Tool & Real-Time Software for Code — Sonar. [Online]. Available: https://www.sonarsource.com/products/sonarlint/

[36] ——. (2025) Build better software, faster. [Online]. Available: https://www.sonarsource.com/

[37] ——. (2025) Quality, Security & Static Analysis Tool with SonarQube. [Online]. Available: https://www.sonarsource.com/products/sonarqube/

[38] Bitwarden. (2025). [Online]. Available: https://github.com/bitwarden

[39] K. Goseva-Popstojanova and A. Perhinschi, “On the capability of static code analysis to detect security vulnerabilities,” *Inf. Softw. Technol.*, vol. 68, pp. 18–33, 2015.

[40] S. Nocera, M. Di Penta, R. Francese, S. Romano, and G. Scanniello, “If it’s not sbom, then what? how italian practitioners manage the software supply chain,” in *Proceedings of International Conference on Software Maintenance and Evolution*. IEEE, 2024, pp. 730–740. [Online]. Available: https://doi.org/10.1109/ICSME58944.2024.00077

[41] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in Software Engineering*. Springer, 2012.

[42] S. Nocera, S. Romano, M. D. Penta, R. Francese, and G. Scanniello, “Software bill of materials adoption: A mining study from github,” in *2023 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2023, pp. 39–49. [Online]. Available: https://doi.org/10.1109/ICSME58846.2023.00016

[43] Z. Zhao, Y. Chen, A. A. Bangash, B. Adams, and A. E. Hassan, “An empirical study of challenges in machine learning asset management,” *Empir. Softw. Eng.*, vol. 29, no. 4, p. 98, 2024.

[44] D. Bifolco, S. Nocera, S. Romano, M. Di Penta, R. Francese, and G. Scanniello, “On the accuracy of github’s dependency graph,” in *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering*. Association for Computing Machinery, 2024, p. 242–251. [Online]. Available: https://doi.org/10.1145/3661167.3661175

[45] D. Bifolco, S. Romano, S. Nocera, R. Francese, G. Scanniello, and M. Di Penta, “An empirical study on the accuracy of github’s dependency graph and the nature of its inaccuracy,” *Information and Software Technology*, p. 107854, 2025. [Online]. Available: https://doi.org/10.1016/j.infsof.2025.107854

[46] Sabato Nocera. (2025) Dealing with SonarQube Cloud: Initial Results from a Mining Software Repository Study - Replication Package. [Online]. Available: https://doi.org/10.6084/m9.figshare.29097383