SonarQube - source code analysis tool

What is SonarQube

SonarQube is a web-based open-source platform used to measure and analyse the quality of source code. Its static code analysis provides insights into code issues and technical debt, helping to assess the code quality in a software project, but also to estimate the remaining effort needed for achieving the production level. SonarQube also helps with tracking code coverage with unit tests. These features reduce the chances of deploying broken or untested code, particularly during the maintenance phase. Use of such a tool helps to identify many bugs and vulnerabilities that would otherwise stay undetected and cause damage. SonarQube's tracking of quality norms allows enforcing them and making the code more reliable and readable. Readability for its part increases productivity and quality, as developers must read many lines of code before editing one; therefore, making the code easier to read makes it easier to write.

SonarQube can be used by the development team and in external reviews. It can analyse and manage the code in more than 25 programming languages, including Java, Python, JavaScript, Swift, PHP, C, C++, C#, PL/SQL, Ruby, etc., but also HTML, XML, and CSS. More than 50 plugins extend its functionality.

How it works

SonarQube reads the source code from the repositories or local files*,* analyses them with dedicated scanners, calculates metrics, stores the findings in a database, and shows the results on its web dashboard. The outcomes of the analysis are quality measures and individually detected issues, which are instances where coding rules were broken. It can analyse source code in several ways:

- On-demand:
    - The provided software may be a locally stored source code.
    - Code is stored in a Git, Bitbucket or SVN source code repository (including GÉANT Gitlab and Bitbucket) and code used by a build management system such as Maven, Gradle, MSBuild or makefile.
- The recommended scenario: continuous inspection within a continuous integration and deployment integration (DevOps) lifecycle managed by a GÉANT tool such as:
    - GitLab (GitLab info)
    - Bamboo (Bamboo info)
    - Bitbucket (Bitbucket info)
    - Jenkins (Jenkins info)

The dashboard incorporates various gauges, grading scales, views, and reports and allows drilling into individual statistics to see exactly why things are flagged up to the causing line of code. This line is displayed with the corresponding issue and suggestion, but also the issue type, severity, status, and time when it was detected as well as the estimated effort.

One instance of SonarQube can support multiple projects, each combining several programming and content language.

Besides using SonarQube as a standalone server, one can use SonarCloud instance provided by SonarSource. It is free for open-source projects and can be easily integrated with projects on GitHub, Bitbucket provided by Atlassian and Azure DevOps server. GN software teams should use its SonarQube inst ance, but this option makes it easier to use it in broader collaborations.

Developers may integrate their IDE (Integrated Development Environment such as Eclipse, IntelliJ, NetBeans, MS Visual Studio) with SonarQube in order to access the detected issues directly form their programming environment.

SonarQube is distributed under the GNU LGPL license version 3. It is maintained by SonarSource.

What it provides

SonarQube uses the software version, time or date defined period to identify the new code. The new code typically introduces the new problems, particularly if the previously written code has been in production and was pruned for errors by more extensive testing, usage, and maintenance. The new code perspective allows the developers to focus on the code they add or change, instead of looking at the debt that is already in the system and thus quickly spot and early fixed new issues.

The list of projects allows comparing the key quality characteristics across multiple projects or components. In addition to allowing focusing on overall status or the new code, the specific perspectives allow to simultaneously observe a set of project metrics associated with a specific concern:

- Risk – reliability and security ratings, test coverage, technical debt, and lines of code.
- Reliability – reliability rating, reliability remediation effort, lines of code, and bug count.
- Security – security rating, security remediation effort, lines of code, and vulnerability count.
- Maintainability – maintainability rating, technical debt, lines of code, and code smell count.
- Coverage – coverage, complexity, and uncovered lines.
- Duplications – duplicated lines %, lines of code, and duplicated blocks.

Each of these visualizations is displayed with time on X-axis. Such graphs allow tracking trends. The same measurement comparisons and related visualizations can be used for project components at the project level.

An application may work without bugs and still be difficult to manage and become unstable after any change. Metrics on code smell and code coverage related to technical debt address these concerns by refactoring the code before it becomes unmaintainable. In SQ, the technical debt is expressed as the estimated time required to fix all maintainability issues and code smells. Maintainability is based on the number of code smells, i.e. suspicious places in the code that indicate possible weakness in design or readability, technical debt, i.e. the effort to fix all code smells, estimated in minutes or workdays, or technical debt ratio, which is the ratio between the cost to develop the software and the cost to fix it, based on the time cost of the issues and the estimate of the time to write the given number of lines of code. Many other provided measures are linked with associated effort, which is the estimated time needed to address them.

SonarQube analysers contribute rules which are executed on source code to generate issues. They use the most advanced techniques, such as pattern matching and dataflow analysis, to analyse the code. For example, there are 547 rules for Java, 210 rules for JavaScript, 186 rules for PO/SQL, 56 rules for HTML, and 24 rules for CSS (https://rules.sonarsource.com/). There are four types of rules:

- Bugs (reliability-related issues) indicate that there something wrong in the code, even if the code currently works, it is broken and needs to be fixed.
- Code smells (maintainability related) are certain common patterns in the code that indicates that the code in question does not satisfy the basic design, implementation and quality principles that may slow down the development or increase the risks.
- Vulnerabilities (security-related) are issues that most likely introduce security risks. Covered vulnerabilities include those included in OWASP Top 10 Application Security Risks and SANS Top 25 Most Dangerous Software Errors.
- Security hotspots (security-related) draw attention to the pieces of code that use security-sensitive APIs (that use weak algorithms, connect to a database, etc.). These hotspots must be manually reviewed to determine whether they introduce vulnerabilities.

A quality profile is a set of rules used by SonarQube to classify and describe issues. A snapshot is a set of measures and issues on a given project at a given time, generated during each analysis. Each snapshot is based on a single quality profile.

Quality gates are an instrument to set a policy for shipping code to production. They are a synthetic measure which determines whether a project has passed or failed. The set of requirements set by gates tells whether or not a new version of a project can go into testing or production. These gates set controls on what is deployed to production without any additional manual effort. They may be customized in line with needed levels of specific quality characteristics that are relevant for the particular software. They guarantee that code added and changed will meet the quality requirement and that overall quality of the application increases from version to version. This mechanism allows integrating static code analysis quality gates into continuous integration pipeline.

In addition, depending on project needs, specific quality control rules can be disabled or enabled. For code smells and bugs, zero false-positives are expected. For vulnerabilities, the target is to have more than 80% of the issues to be true-positives. For security hotspot rules, it is expected that more than 80% of the issues will be quickly resolved as "Won't Fix" after review by a security auditor.

All described features allow exploring and highlighting the critical areas for improvement, as required by on the context of the project.