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, and also to estimate the remaining effort needed for achieving the production level. SonarQube also supports reporting the quality with regard to unit tests through reporting a code coverage percentage. The SonarQube features reduce the chances of deploying broken or untested code, particularly during the maintenance phase. The usage of such a tool helps to identify many bugs and vulnerabilities that would otherwise stay undetected and could 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 and modify.

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 does SonarQube work ?

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 the coding rules were broken. It can analyse source code in several ways:

- On-demand: the software is inspected once and after any change, a re-run of a SonarQube analysis must be triggered manually;
- Automated: the automation of software reviews is considered a recommended option. The continuous inspection is implemented by connecting the source code repositories with SonarQube. Helper processes (e.g., GitLab runner) trigger the start of code analysis in an automated way, e.g., by a commit message of a code change by the developer team;

SonarQube's dashboard incorporates various gauges, grading scales, views, or 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 languages.

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

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

What does SonarQube provide?

SonarQube uses the software version, time- or date-defined period to identify the new code. The new code typically introduces new problems, particularly if the previously written code has been in production and was pruned of 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 technical debt that is already in the system, and thus quickly spot and early fix new issues.

The list of projects allows for comparing the key quality characteristics across multiple projects or components. In addition to allowing focusing on the overall status or the state of 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 visualisations is displayed with time on X-axis. Such graphs allow tracking trends. The same measurement comparisons and related visualisations can be used for project components at the project level.

An application may work without bugs, while still being difficult to manage and becoming unstable after any change. Metrics on code smell and code coverage related to technical debt address these concerns by refactoring the code before it actually becomes unmaintainable. In SonarQube, the technical debt is expressed as the estimated time required to fix all maintainability issues and code smells. Maintainability is an aggregate value based on several metrics: the number of code smells, i.e. suspicious places in the code that indicate possible weakness in design or readability, a spot of technical debt, or a technical debt ratio, which is the ratio between the cost to develop the software and the cost of removing the accumulated debt, 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 PL/SQL, 56 rules for HTML, and 24 rules for CSS (https://rules.sonarsource.com/). There are four types of rules:

- Code smells (maintainability related) are certain common patterns in the code that indicate 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.
- Bugs (reliability-related issues) indicate that there is something wrong in the code, even if the code currently seems to work correctly, it is broken and needs to be fixed.
- 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.). While hotspots are not confirmed vulnerabilities, they indicate potential issues and must be manually reviewed to determine whether they pose risk to the application.

A quality profile is a set of selected rules used by SonarQube to classify and describe issues. They allow for customising the rules to specific user needs. 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 instruments to set a policy for shipping the code to production. They are a synthetic measure that determines whether a project has passed or failed for a pre-defined set of criteria. 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, as they can be triggered manually. They may be customised in line with the needed levels of specific quality characteristics that are relevant for the particular software. They guarantee that the code added and changed meets the quality requirements and that the overall quality of the application is properly monitored from version to version.

The default Quality Gate expects zero false positives code smells and bugs. For vulnerabilities, the target is to have more than 80% of the issues as 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 a review by a security auditor. All thresholds of the default Quality Gate can be adjusted per project needs by simply enabling or disabling specific quality control rules. In this sense, each project can work with its dedicated Quality Gate to optimally reach its specific project goals.

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