What is **SonarQube**?

[SonarQube](http://www.sonarqube.org/)® software (previously called Sonar) is an open source quality management platform, dedicated to continuously analyze and measure technical quality, from project portfolio to method.

[SonarQube](http://www.sonarqube.org/) is a central server that processes full analyses (triggered by the various SonarQube Scanners). Its purpose is to give a 360° vision of the quality of the code base. To achieve this, it analyzes all the source lines of your project on a regular basis.

SonarQube collects and analyzes source code, measuring quality and providing reports for your projects. It combines static and dynamic analysis tools and enables quality to be measured continuously over time. Everything that affects our code base, from minor styling details to critical design errors, is inspected and evaluated by SonarQube, thereby enabling developers to access and track code analysis data ranging from styling errors, potential bugs, and code defects to design inefficiencies, code duplication, lack of test coverage, and excess complexity.

The Sonar platform analyzes source code from different aspects and hence it drills down to your code layer by layer, moving from the module level down to the class level. At each level, SonarQube produces metric values and statistics, revealing problematic areas in the source that require inspection or improvement

Features and Scope

SonarQube covers the code quality in below stages: (Seven Axes of Quality)

|  |  |  |  |
| --- | --- | --- | --- |
| Architecture & Design | Comments | Coding rules | Potential Bugs |
| Duplications | Unit Tests | Complexity | |

SonarQube can perform analysis on 25+ different languages. The outcome of this analysis will be quality measures and issues (instances where coding rules were broken). However, what gets analyzed will vary depending on the language

1. On all languages, a static analysis of source code is performed (Java files, COBOL programs, etc.)
2. A static analysis of compiled code can be performed for certain languages (.CLASS files or jars in Java, .DLL files in C#, etc.)
3. A dynamic analysis of code can be performed on certain languages (execution of unit tests in Java, C#, etc.)
4. SonarQube is also used for Android Development.
5. SonarQube internally uses PMD, Findbugs, CheckStyle etc.
6. Integration of SonarQube with standard ALM components such as Maven, Ant, SVN, Git, Mercurial, JIRA, Mantis, Google Analytics, Piwik, Fortify etc. comes out of the box.

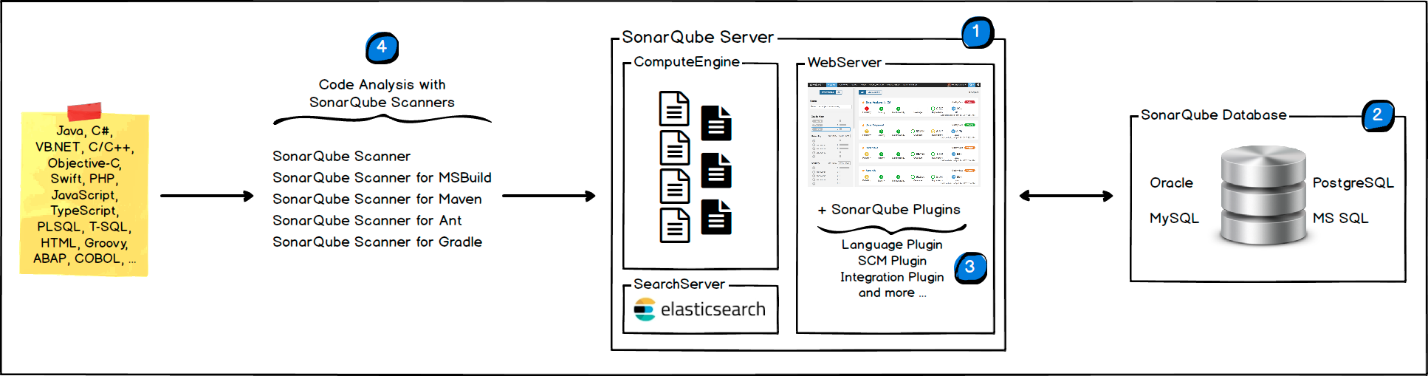
Why SonarQube

* SonarQube doesn't just show you what's wrong. It also offers quality-management tools to actively help you put it right
* SonarQube's commercial competitors seem to focus their definition of quality mainly on bugs and complexity, whereas SonarQube's offerings span what its creators call the Seven Axes of Quality
* SonarQube addresses not just bugs but also coding rules, test coverage, duplications, API documentation, complexity, and architecture, providing all these details in a dashboard
* It gives you a moment-in-time snapshot of your code quality today, as well as trends of lagging (what's already gone wrong) and leading (what's likely to go wrong in the future) quality indicators
* It provides you with metrics to help you take right decision. In nearly every industry, serious leaders track metrics. Whether it's manufacturing defects and waste, sales and revenue, or baseball hits and RBIs, there are metrics that tell you how you're doing: if you're doing well overall, or whether you're getting better or worse.

Static analysis is performed in a non-runtime environment. Typically, a static analysis tool will inspect program code for all possible run-time behaviors and seek out coding flaws, back doors, and potentially malicious code.

Dynamic analysis adopts the opposite approach and is executed while a program is in operation. A dynamic test will monitor system memory, functional behavior, response time, and overall performance of the system.

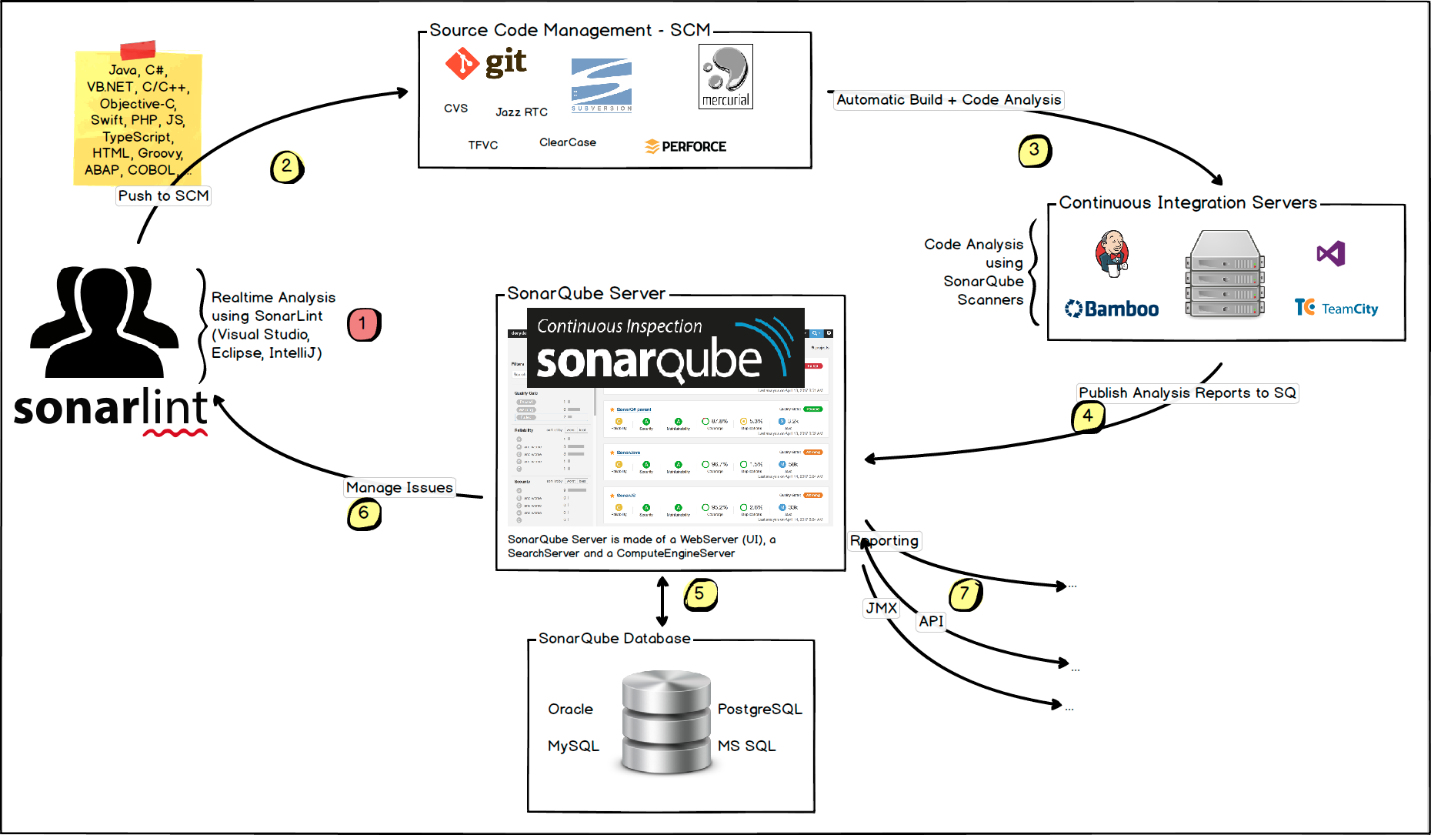
Architecture



The SonarQube Platform is made of 4 components:

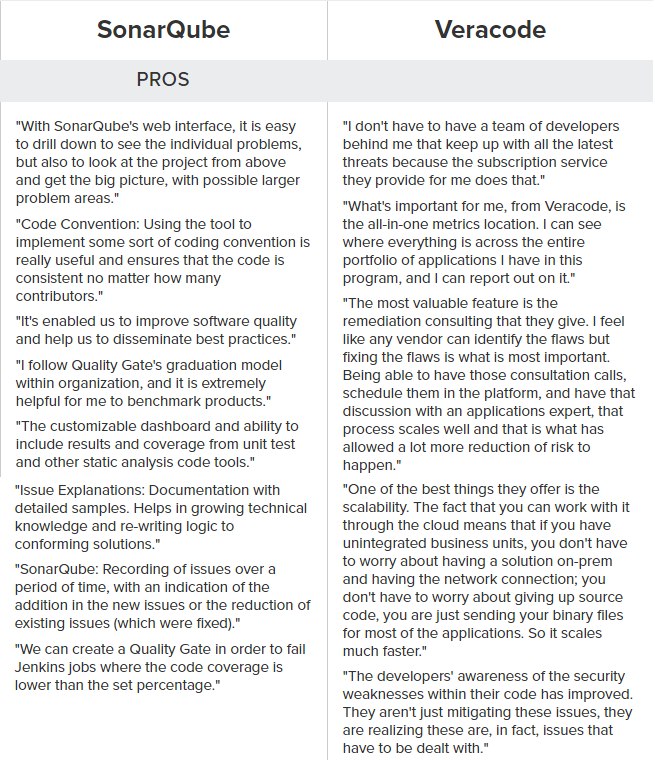
1. One SonarQube Server starting 3 main processes
   * Web Server for developers, managers to browse quality snapshots and configure the SonarQube instance
   * Search Server based on Elasticsearch to back searches from the UI
   * Compute Engine Server in charge of processing code analysis reports and saving them in the SonarQube Database
2. One SonarQube Database to store:
   * the configuration of the SonarQube instance (security, plugins settings, etc.)
   * the quality snapshots of projects, views, etc.
3. Multiple SonarQube Plugins installed on the server, possibly including language, SCM, integration, authentication, and governance plugins
4. One or more SonarQube Scanners running on your Build / Continuous Integration Servers to analyze projects

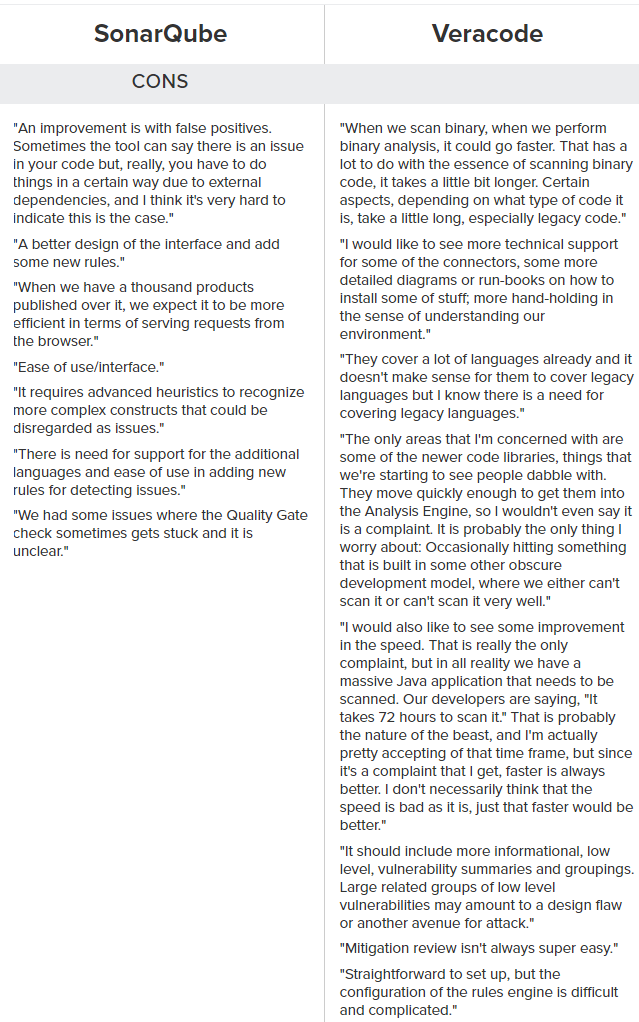
Integration



The following schema shows how SonarQube integrates with other ALM tools and where the various components of SonarQube are used.

* Developers code in their IDEs and use [SonarLint](http://www.sonarlint.org/) to run local analysis.
* Developers push their code into their favorite SCM: git, SVN, TFVC,
* The Continuous Integration Server triggers an automatic build, and the execution of the SonarQube Scanner required to run the SonarQube analysis.
* The analysis report is sent to the SonarQube Server for processing.
* SonarQube Server processes and stores the analysis report results in the SonarQube Database, and displays the results in the UI.
* Developers review, comment, challenge their Issues to manage and reduce their Technical Debt through the SonarQube UI.
* Managers receive Reports from the analysis.  
  Ops use APIs to automate configuration and extract data from SonarQube.  
  Ops use JMX to monitor SonarQube Server.

VeraCode Vs SonarQube (User Reviews) ****

****