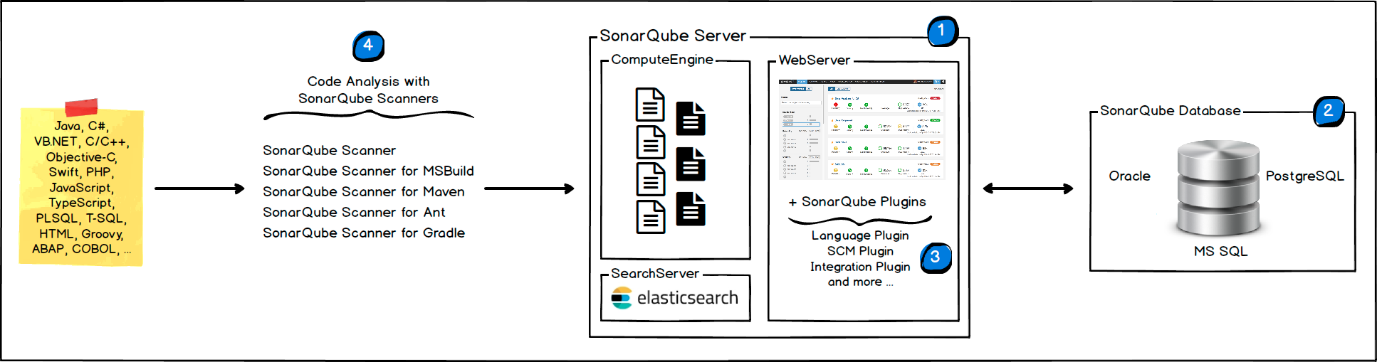
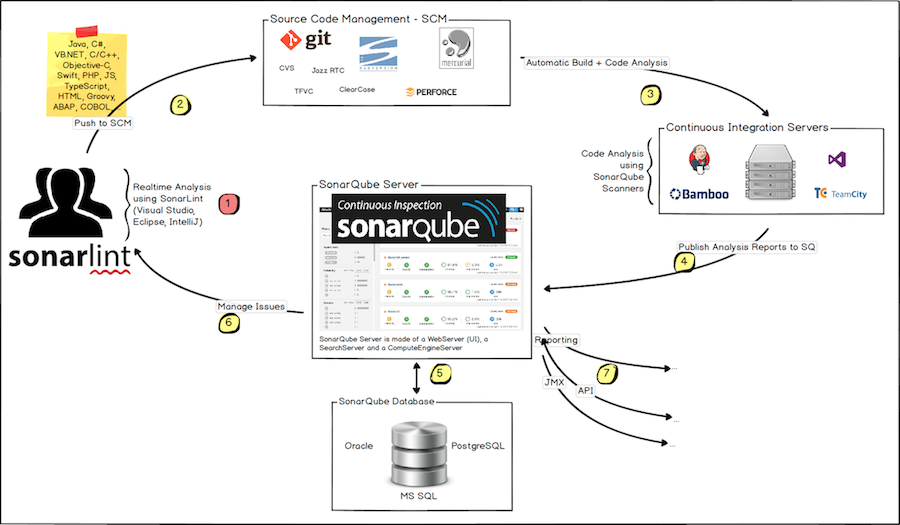
[SonarQube](http://www.sonarqube.org/" \t "_blank)® is an automatic code review tool to detect bugs, vulnerabilities and code smells in your code.



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


1. Developers code in their IDEs and use [SonarLint](https://sonarlint.org/" \t "_blank) to run local analysis.
2. Developers push their code into their favourite SCM : git, SVN, TFVC, ...
3. The Continuous Integration Server triggers an automatic build, and the execution of the SonarScanner required to run the SonarQube analysis.
4. The analysis report is sent to the SonarQube Server for processing.
5. SonarQube Server processes and stores the analysis report results in the SonarQube Database, and displays the results in the UI.
6. Developers review, comment, challenge their Issues to manage and reduce their Technical Debt through the SonarQube UI.
7. Managers receive Reports from the analysis. Ops use APIs to automate configuration and extract data from SonarQube. Ops use JMX to monitor SonarQube Server.

| **Concept** | **Definition** |
| --- | --- |
| Bug | An issue that represents something wrong in the code. If this has not broken yet, it will, and probably at the worst possible moment. This needs to be fixed. Yesterday. |
| Code Smell | A maintainability-related issue in the code. Leaving it as-is means that at best maintainers will have a harder time than they should making changes to the code. At worst, they'll be so confused by the state of the code that they'll introduce additional errors as they make changes. |
| Cost | See Remediation Cost |
| Debt | See Technical Debt |
| Issue | When a piece of code does not comply with a rule, an issue is logged on the **snapshot**. An issue can be logged on a source file or a unit test file. There are 3 types of issue: **Bugs**, **Code Smells** and **Vulnerabilities** |
| Measure | The value of a **metric** for a given file or project at a given time. For example, 125 lines of code on class MyClass or density of duplicated lines of 30.5% on project myProject |
| Metric | A type of measurement. Metrics can have varying values, or **measures**, over time. Examples: number of lines of code, complexity, etc. A metric may be either *qualitative* (gives a quality indication on the component, E.G. density of duplicated lines, line coverage by tests, etc.) or *quantitative* (does not give a quality indication on the component, E.G. number of lines of code, complexity, etc.) |
| New Code Period | The period for which you're keeping a close watch on the introduction of new problems in the code. Ideally this is since the previous\_version, but if you don't use a Maven-like versioning scheme you may need to set a relatively arbitrary time period such as 21 days or since a specific date. |
| Quality Profile | A set of **rules**. Each **snapshot** is based on a single quality profile. See also [Quality Profiles](https://docs.sonarqube.org/latest/instance-administration/quality-profiles/) |
| Rule | A coding standard or practice which should be followed. Not complying with coding rules leads to **Bugs**, **Vulnerabilities**, **Security Hotspots**, and **Code Smells**. Rules can check quality on code files or unit tests. |
| Remediation Cost | The estimated time required to fix Vulnerability and Reliability Issues. |
| Snapshot | A set of **measures** and **issues** on a given project at a given time. A snapshot is generated for each analysis. |
| Security Hotspot | Security-sensitive pieces of code that need to be manually reviewed. Upon review, you'll either find that there is no threat or that there is vulnerable code that needs to be fixed. |
| Technical Debt | The estimated time required to fix all Maintainability Issues / code smells |
| Vulnerability | A security-related issue which represents a backdoor for attackers |