Glossary of technical terms

# Blocker

In the context of code analysis, blocker severity is often used to refer to the most critical or severe issues that prevent the code or system from functioning as intended. These issues can cause the system to crash or prevent the application from launching, and they should be addressed as soon as possible.

Fixing blocker severity issues is critical to the functionality of the code and they should be given the highest priority in terms of remediation.

It's important to note that the definition of blocker severity can vary depending on the context of the project, the team and the specific requirements. It can also be considered as a subset of high severity issues and sometimes be used interchangeably with it.

# Bug

In the context of code analysis, a bug is an error or defect in the code that causes it to behave in unintended ways. Bugs can manifest in a variety of ways, such as causing the program to crash, producing incorrect or unexpected results, or causing unexpected behavior.

The goal of code analysis is to identify and fix bugs before they can cause problems in production. This can help improve the quality and reliability of the code and reduce the risk of security vulnerabilities.

It's important to note that code analysis is not a guarantee that a codebase is bug-free and some bugs may only be discovered during testing or in production.

# Code Smell

Code smell is a term used to describe certain characteristics of code that may indicate a deeper problem. It is not a defect or an error, but it can be a sign that the code is difficult to understand, maintain, or extend. Code smells can be caused by poor design, lack of proper structure, or lack of adherence to coding standards.

Code smells are not always a problem, but they can be a sign that the code needs to be refactored or redesigned to improve its maintainability and readability.

It's important to note that detecting code smell is a subjective process and its interpretation may vary depending on the context, the team and the project.

# Critical

In the context of code analysis, critical severity is often used to refer to the most severe issues that have a significant impact on the code or the system, and can cause security vulnerabilities, data loss or system failure. These issues should be addressed as soon as possible.

Fixing critical severity issues is critical to the security and reliability of the code, and they should be given the highest priority in terms of remediation.

It's important to note that the definition of critical severity can vary depending on the context of the project, the team and the specific requirements. It can also be considered as a subset of high severity issues and sometimes be used interchangeably with it.

# Maintainability

Maintainability is a measure of how easy it is to understand, modify, and extend a piece of code over time. A maintainable codebase is one that can be easily understood and modified by developers, which helps to reduce the likelihood of bugs and improves overall efficiency.

Maintaining a codebase is an ongoing process, and developers should continuously monitor the codebase and refactor it as needed to keep it maintainable.

In Sonarqube the issue *Code Smell* decides the maintainability rating.

# Major

In the context of code analysis, major severity is often used to refer to the most critical or severe issues that have a significant impact on the code or the system. These issues can cause security vulnerabilities, data loss, or system failure, and they should be addressed as soon as possible.

Fixing major severity issues is critical to the security and reliability of the code, and they should be given the highest priority in terms of remediation.

It's important to note that the definition of major severity can vary depending on the context of the project, the team and the specific requirements.

# Refactoring

Refactoring is the process of modifying existing code to improve its quality, maintainability, or performance without changing its external behavior. It involves restructuring the codebase to make it easier to understand, maintain, and extend.

Refactoring should be done carefully and incrementally, testing the code after each change to ensure that the external behavior of the code remains the same.

# Reliability

Reliability refers to the ability of a piece of code to perform its intended function consistently and without errors. A reliable codebase is one that is free of bugs and can be trusted to work as expected.

Reliability is an essential aspect of code quality, and developers should take steps to ensure that their code is reliable, by testing and validating it, monitoring and logging errors and exceptions, and continuously refactoring it to improve its design and performance.

In Sonarqube the issue *Bug* decides the reliability rating.

# Security

Security is the practice of protecting code and systems from unauthorized access, use, disclosure, disruption, modification, or destruction. A secure codebase is one that is protected against known and unknown vulnerabilities and can be trusted to keep sensitive information safe.

Keeping software secure is a continuous effort and requires developers to stay up to date with the latest security best practices, and continuously monitor and update the codebase to address new vulnerabilities.

In Sonarqube the issue *Vulnerability* decides the security rating.

# Severity

In the context of code analysis, severity refers to the level of impact that a particular issue or bug has on the code or the system. Different coding standards or frameworks may use different levels of severity, but generally, they are classified into four main categories:

* High Severity: These are the most critical issues and have a significant impact on the code or the system. They can cause security vulnerabilities, data loss, or system failure.
* Medium Severity: These are issues that have a moderate impact on the code or the system. They may cause performance issues, unexpected behavior, or usability problems.
* Low Severity: These are issues that have a minor impact on the code or the system. They may be cosmetic issues, such as code formatting problems, or minor coding errors that do not affect the functionality of the system.
* Informational: These are issues that do not have any impact on the code or the system. They are typically best practices or guidelines to improve the codebase's readability, maintainability or performance.

In general, High severity issues should be prioritized and fixed as soon as possible, while low severity issues can be addressed in a future release or sprint.

It's important to note that the severity of an issue can depend on the context of the project and the team, and may vary depending on the use case, the environment, and the specific requirements.

# Vulnerability

A vulnerability in code analysis refers to a weakness or flaw in the code analysis process that can lead to false positives, false negatives, or other inaccuracies in the analysis results. This can lead to issues being missed or incorrectly identified, which can have serious consequences if the issues are actually vulnerabilities.

It's important to perform regular code analysis and to use multiple tools and techniques to ensure that vulnerabilities are accurately identified. Additionally, it's important to keep the code analysis tools and rules updated to ensure that they can detect the latest vulnerabilities and to have a team with security expertise to review the results.

It's also important to have transparency and accountability in the code analysis process and to have a regular review of the analysis results by the team and stakeholders.