Sonar

The State of Code

Volume 3: Maintainability

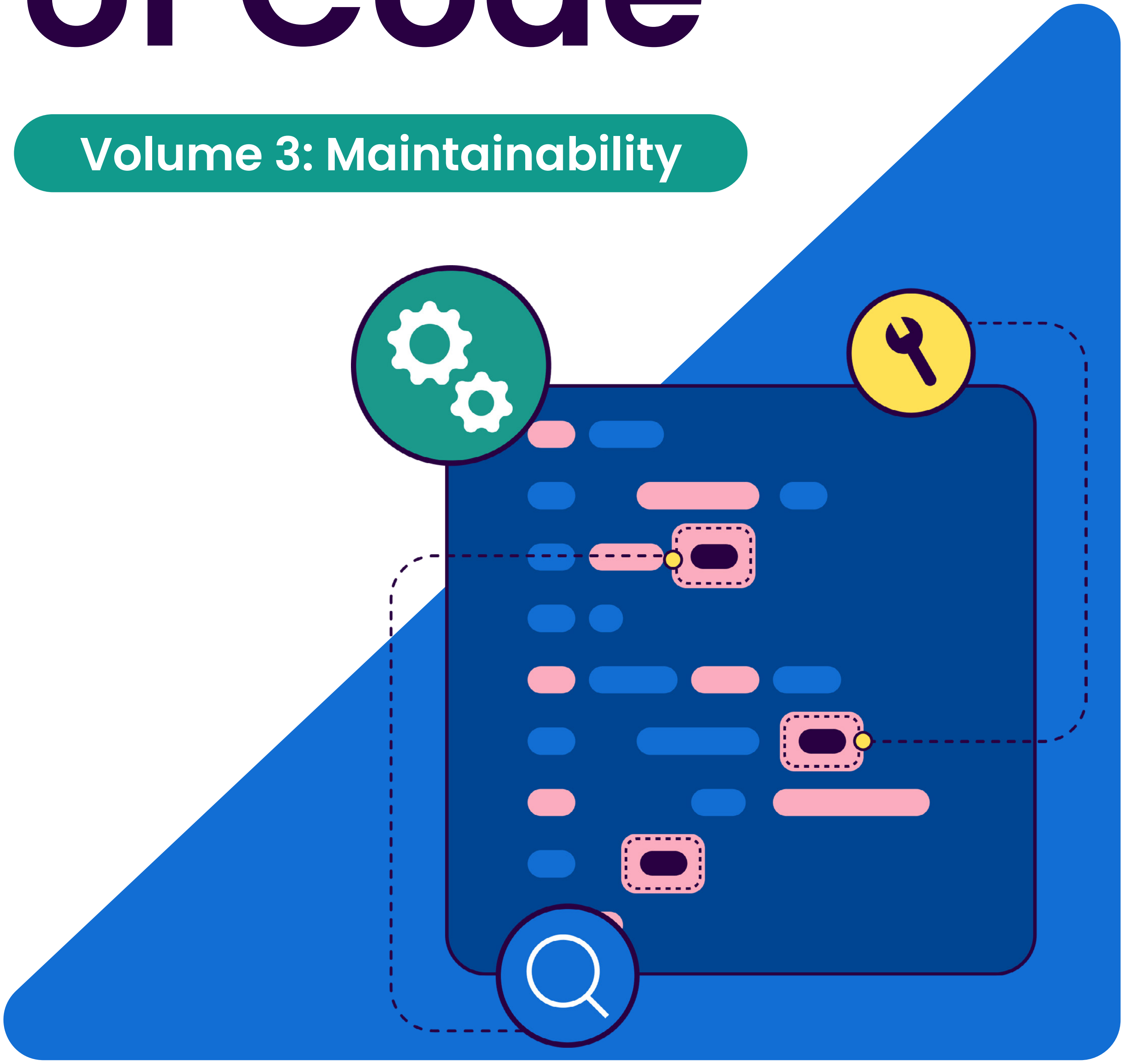


Table of Contents

Introduction

Sonar's integrated solutions for code security and code quality, called SonarQube, examine 300 billion lines of code every single day in order to help development teams ship high-quality, secure code. Given our unique visibility into the code developers are creating today, we analyzed a subset of that code in order to extract the top issues that surfaced through our analysis. This is the third in a series of reports highlighting our findings.

In this report, we highlight the most common code maintainability issues (sometimes called code smells) that can affect software quality. This context is essential for software developers, their leaders, and AppSec stakeholders who need to make (or justify) software engineering investment decisions around training, tooling, or technical debt and would benefit from knowing what issues may be lurking or unknown in their critical software. And as AI coding assistants generate more code, the quality of existing application source code becomes more important, as it is the main data used to train these AI tools.

For background, Sonar measures the impact of code issues in every project or codebase across three software qualities: reliability, security, and maintainability. These three areas are deeply interconnected in high-quality code: poorly maintained code typically develops reliability issues and security vulnerabilities over time. Taken together, reliability, security, and maintainability determine not just the initial success of software but its long-term value, adaptability, and total cost of software ownership throughout its lifecycle. This is why every rule violation identified by Sonar is automatically assigned an impact quality for one of these three areas.

We focus primarily on the code maintainability issues in this report. Our prior reports, "The State of Code: Reliability" and "The State of Code: Security" are also worth a read. A future report will include common issues broken down by programming language.

Overview and summary of key findings

Software plays a vital role in modern business, increasingly serving as a key differentiator and driver of customer experience. Consequently, the quality of internally developed software will have a growing impact on both customer satisfaction and competitive standing.

Poor software quality, therefore, represents a significant threat to every business. Recent projections estimate that the annual cost of poor software quality in the US has risen to over $2.41 trillion (Consortium for Information & Software Quality, 2022).

The staggering growth of new code that is generated or touched by AI tools dramatically increases the impact of this trend. AI coding tools are excellent mimics: they create based on what they learn from existing human code. It follows that code quality problems that commonly exist today will continue to pop up in AI generated code, and these issues must be addressed.

This report includes the most common maintainability issues (alternatively called code smells) impacting our users' software. Sonar's analysis exposed maintainability issues like improper variable declarations and unit tests lacking assertions that could cause severe consequences for applications running in production if not addressed.

Key findings:

- On average, every million lines of code Sonar analyzed contained about 53,000 maintainability issues.
- Said differently, Sonar caught 72 code smells per developer per month during the period examined.
- The most frequently found code smell relates to JavaScript variables declared with overly broad scope, which could lead to hard-to-find bugs and unpredictable behavior.
- The most frequent blocker code smell involved test methods that lacked assertions, which can indicate tests that don't properly verify the code's behavior.

Our methodology

Sonar operates the largest SaaS solution for integrated code security and code quality analysis, so we have a unique view into how code is being written across many languages, industries, organization sizes, and geographies. We looked at SonarQube analysis data for the last six months of 2024, across seven of the most common languages developers use. Unlike other reports that rely on surveys, this analysis is backed by concrete data that shows the real issues developers are encountering as they code. Moreover, each of these data points links back to an issue that was caught by SonarQube and surfaced to developers.

About our dataset

The dataset for this report comes from SonarQube Cloud, which is a SaaS code analysis solution designed to detect coding issues in over 35 languages, frameworks, and Infrastructure-as-Code (IaC) platforms. It integrates directly with continuous integration (CI/CD) pipelines and DevOps platforms (like GitHub, GitLab, Azure DevOps, and Bitbucket), and checks code against an extensive set of rules covering maintainability, reliability, and security issues on each merge/pull request or branch commit. Our analysis evaluates code health against thousands of criteria that have been created by developers, for developers.

For this study, we've included every pull request and branch analysis in 2024 between July 1st and December 31st, where the code was written in **Java, JavaScript, TypeScript, Python, C#, C++, or PHP** (the top software development languages used with SonarQube).

This scope yields a vast dataset encompassing:

- More than **7.9 billion lines of code**.
- Work from over **970,000 developers** across over **40,000 organizations** globally.
- Roughly **445 million code issues** across **5,300 unique quality and security rules**.

Maintainability report findings

Maintainability issues, often referred to as code smells, are problems in the code that affect its maintainability but not necessarily its functionality. They indicate weaknesses in the code design that can make the code hard to read, understand, and maintain, and they can increase the risk of bugs and errors over time.

The most common maintainability issues

Of the approximately 445 million issues identified by Sonar, about 420 million of them were categorized as code smells.

☆ Sonar's analysis uncovered about 53,000 code smells per million lines of code analyzed.

Variables should be declared with "let" or "const"

Languages: JS TS **Volume:** □□□□□□

Variables declared with `var` have function scope and can be accessed within the entire function. `let` and `const` have block scope and are limited to the block of code in which they are defined. `const` variables are also immutable. `let` and `const` are preferred due to their more precise variable types.

String literals should not be duplicated

Languages: C# Java JS TS Python php **Volume:** □□□

Duplicated string literals make refactoring complex and error prone, as any change would need to be propagated on all occurrences. Replace them with constants that can be referenced from many places, but only need to be updated once.

React components should validate prop types

Languages: Java **Volume:** □□□

In JavaScript, props can be passed as plain objects, which can lead to errors. By defining types for component props, developers can enforce type safety and provide clear documentation, which helps catch errors and improve code maintainability.

Fields that are only assigned in the constructor should be "readonly"

Languages: **Volume:** □□□

Fields that are only assigned a value within a class constructor but are not marked as readonly can lead to confusion about their intended use. To prevent this ambiguity and avoid unintentional modifications by future maintainers, developers should explicitly mark these fields as readonly.

Local variable and method parameter names should comply with a naming convention

Languages: **Volume:** □□

Adhering to naming conventions for code elements improves readability and maintainability. Inconsistent naming can lead to errors and collaboration difficulties. To fix this, developers should understand the project's naming convention and update the names accordingly.

Why you should make sure JavaScript variables are declared correctly

The most frequently flagged maintainability issue in our dataset affects variable declarations using ‘var’ in JavaScript instead of ‘let’ or ‘const’. ‘var’ variables exist throughout their entire function, not just within the specific block of code where they were defined. This means that variables can be accessed before they're properly set up (giving mysterious undefined values) and can leak outside of loops or “if” statements. ‘const’ prevents variables from being changed after creation, making it harder to accidentally reassign values. These issues might seem small, but they lead to bugs that are hard to find, especially in larger programs. That's why modern JavaScript introduced ‘let’ and ‘const’—they create clearer boundaries around where variables exist and whether they can be changed.

Spotlight: Top blocker code smell

SonarQube categorizes code issues by severity, with "blocker" issues indicating a high likelihood of severe unintended consequences. We generally recommend that users address blocker issues immediately.

The most common blocker maintainability issue found by SonarQube involved unit tests that should include assertions.

Tests should include assertions

Languages: **Severity:** BLOCKER

Test methods that lack an assertion only verify that the system under test does not throw any exceptions, without providing any guarantees regarding the code's behavior. Enforcing the presence of assertions ensures comprehensive tests by ensuring that they validate expected behavior.

Tests without assertions are problematic because they create a false sense of security. They appear to "pass" but don't verify anything meaningful about the code's correctness. The code could return completely wrong results, corrupt data, or exhibit unexpected behavior, but the test would still pass as long as no exception is thrown. When code is refactored or modified, these tests won't catch if the intended functionality has been broken—they only catch if the new code introduced crashes.

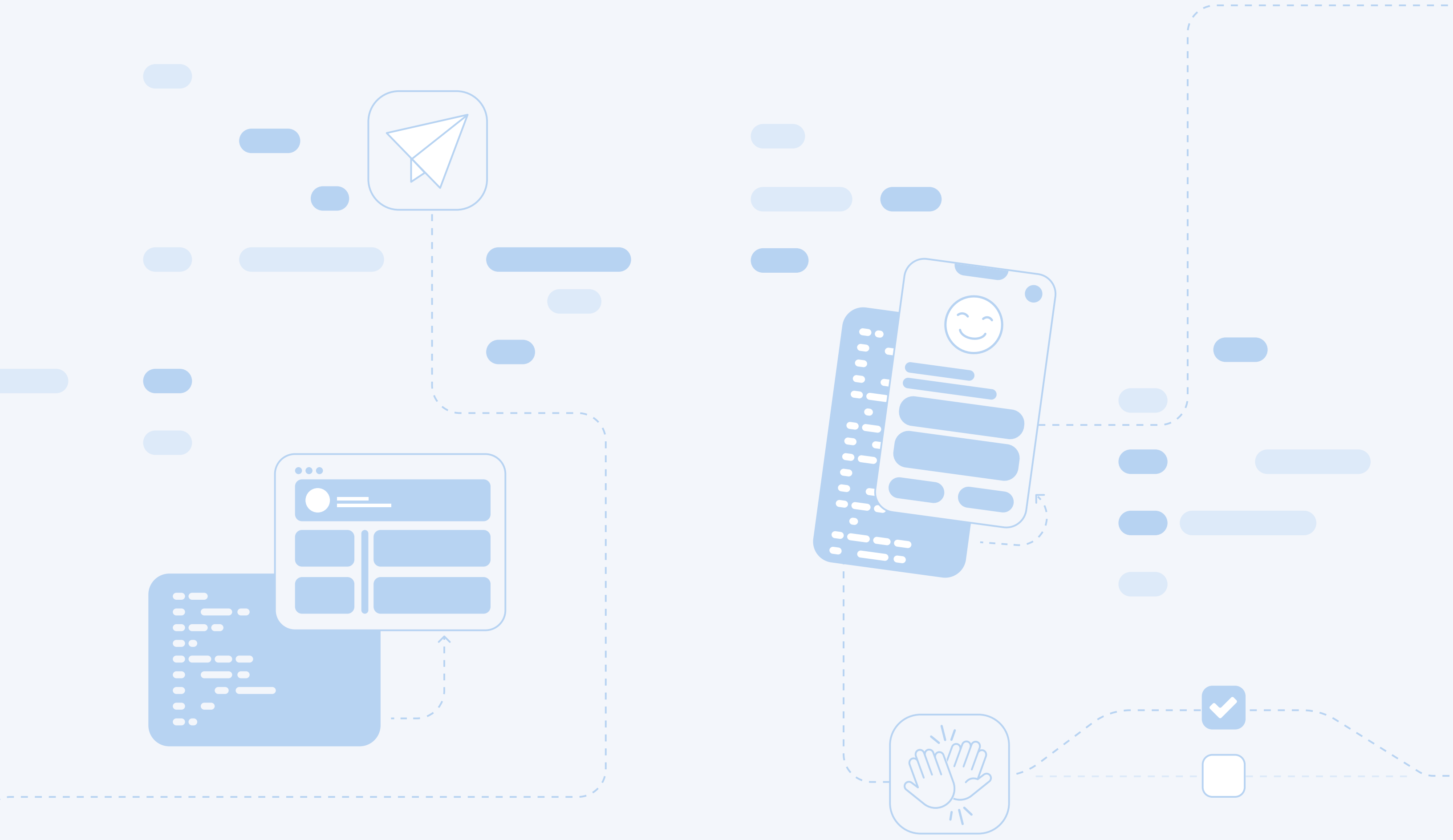
These tests also consume resources (time to run, cognitive load to maintain) while providing no real value in catching bugs or regressions. They inflate test coverage statistics and pass rates without actually improving code quality, making it harder to assess true testing effectiveness. Good tests should verify specific expected outcomes, not just the absence of exceptions. Every test should have a clear assertion that validates the system behaves as intended.

Conclusion

This report is intended to be a first step towards increased transparency around some of the most common maintainability issues that can be found in the source code being actively written and maintained today. It underscores the need for solutions that facilitate automated code review, like SonarQube, to intercept critical issues so they don't escape into production environments.

As our community moves increasingly towards using AI to augment software development and dramatically increase the rate at which new code is created, we think this assessment of the state of code they are analyzing with Sonar will help to highlight frequently-occurring potential failure points and help developers strengthen the value of the code they create.

Sonar



sonar.com