Sonar

The State of Code

Volume 1: Reliability

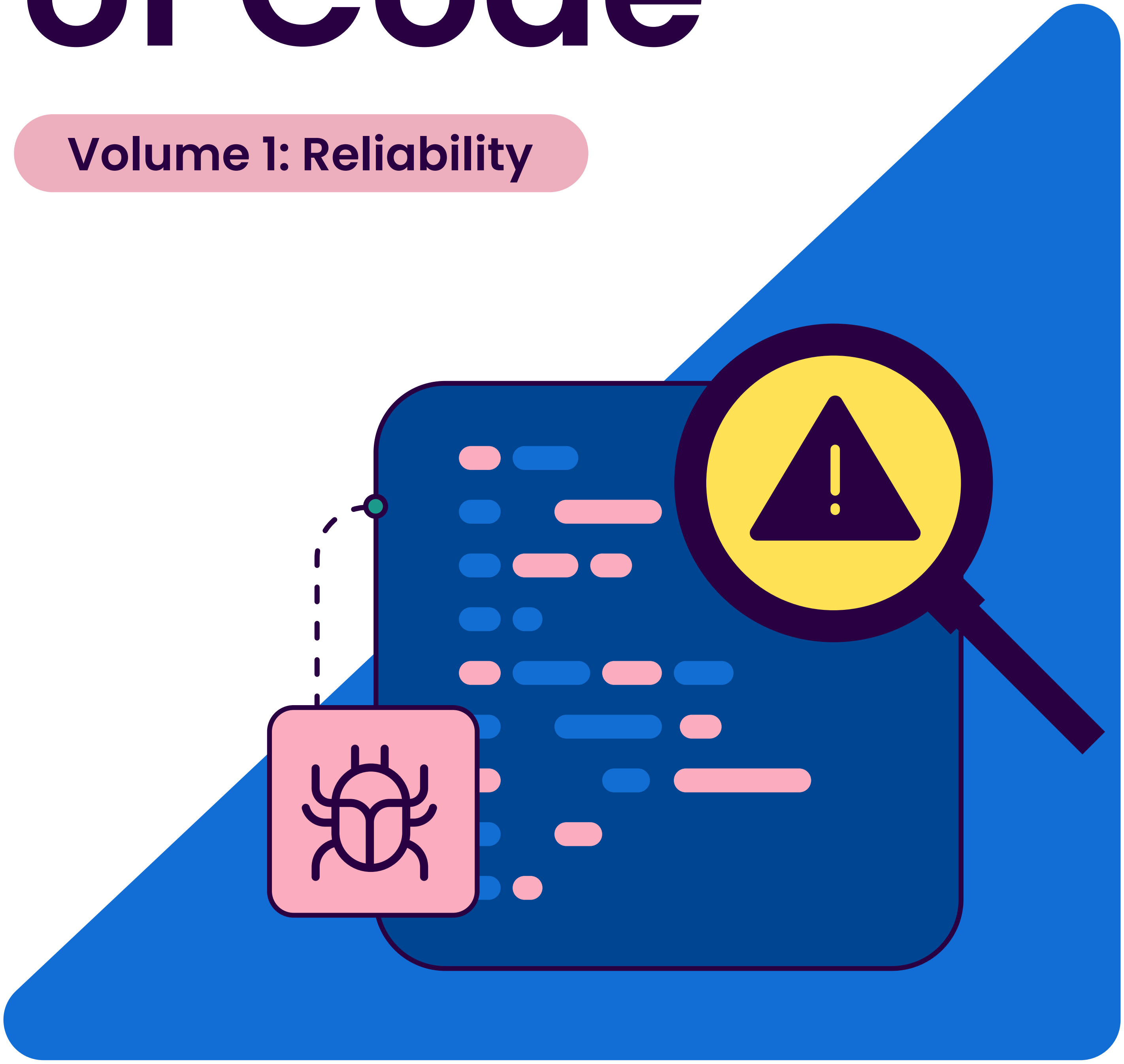


Table of Contents

Introduction 3

Overview and summary of key findings 4

Our methodology 5

Report findings 6

Spotlight: Top blocker bug 8

Conclusion 9

About Sonar 9

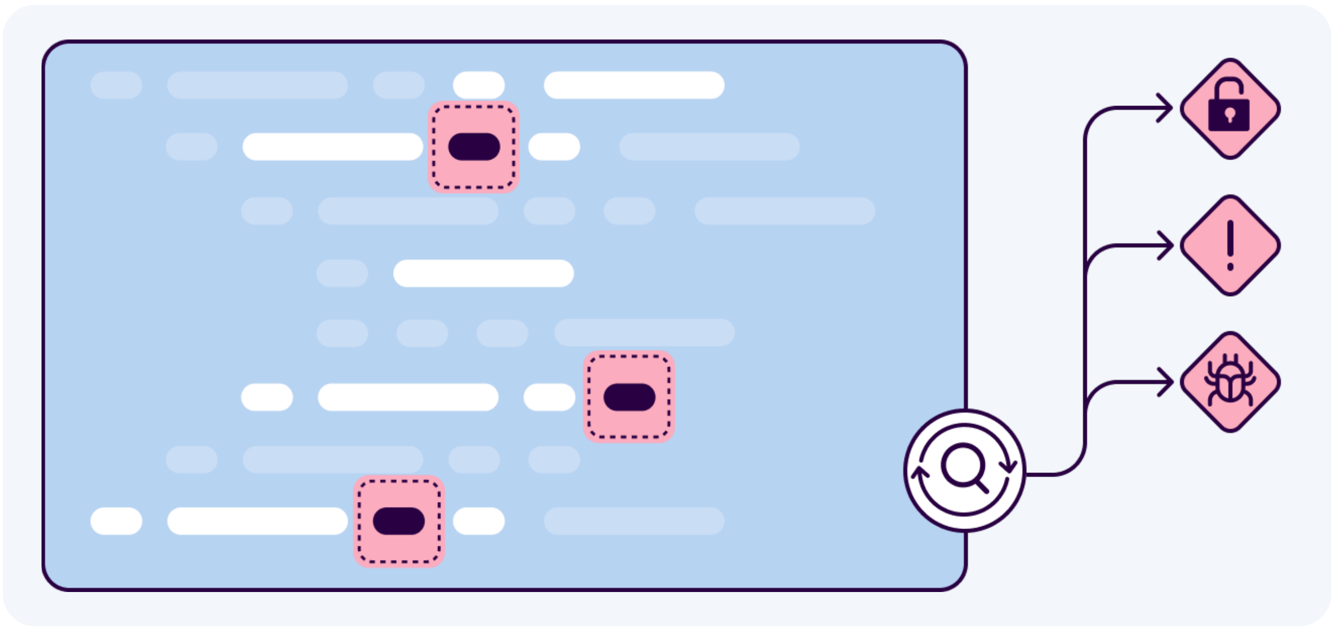
Introduction

Sonar's integrated solutions 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.

Our findings highlight the most common issues that can affect software quality. This context is essential for software development leaders and practitioners who need to make (or justify) software engineering investment decisions around training, tooling, or technical debt and would benefit from knowing what quality 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.

Sonar measures issue impact 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.

Every month, Sonar catches about four reliability and security issues on average per developer, as well as about 72 maintainability issues per developer. In this report, we focus primarily on the reliability issues. Stay tuned for future reports, where we'll cover the other kinds of issues.



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).

This report identifies the most common issues impacting the reliability of our users' software. Our analysis surfaced bugs like dead code, mishandled null pointers, and floating point imprecision that could, if left unaddressed, have devastating consequences for applications running in production.

Key findings:

- On average, every million lines of code Sonar analyzed contained about 2,100 reliability issues (bugs).
- Sonar caught about three reliability issues per developer per month during the period examined.
- The most frequently found reliability issues are dead code and illegal memory access.
- The most frequent **blocker** bug we saw involved unclosed resources in Java (which can cause resource leaks if not properly handled).

Our methodology

We looked at Sonar analysis data for the last six months of 2024, across seven of the most common languages developers use. 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. 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 Sonar 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**.

Report findings

The most common reliability issues

Reliability issues or bugs are issues that could affect the software's capability to maintain its level of performance under promised conditions. Of the approximately 445 million issues flagged by Sonar, about 16 million were categorized as reliability issues or bugs (the others were security and maintainability issues that will be explored in future reports).

Sonar's analysis uncovered about 2,100 bugs per million lines of code analyzed.

Non-empty statements should change control flow or have at least one side-effect

Languages: C++ JS TS Python php **Volume:** □□□□□□□□

Statements that have no side effects or do not change control flow can indicate a programming error or be redundant. They might not behave as intended due to errors or be residual from refactoring. Developers should verify that these statements do not contain bugs or redundancies.

Null pointers should not be dereferenced

Languages: C++ C# Java **Volume:** □□□□□

Accessing a null value can cause an exception, or even program termination that could potentially expose sensitive information. To prevent this, developers should ensure that the variable has a value, or check that it is not null before accessing it.

Floating point numbers should not be tested for equality

Languages: C++ C# Java Python **Volume:** □

Floating point numbers are imprecise due to their binary approximation and non-associative arithmetic, so equality checks can be unreliable. Developers should use a tolerance value (epsilon) for comparisons instead.

Mouse events should have corresponding keyboard events

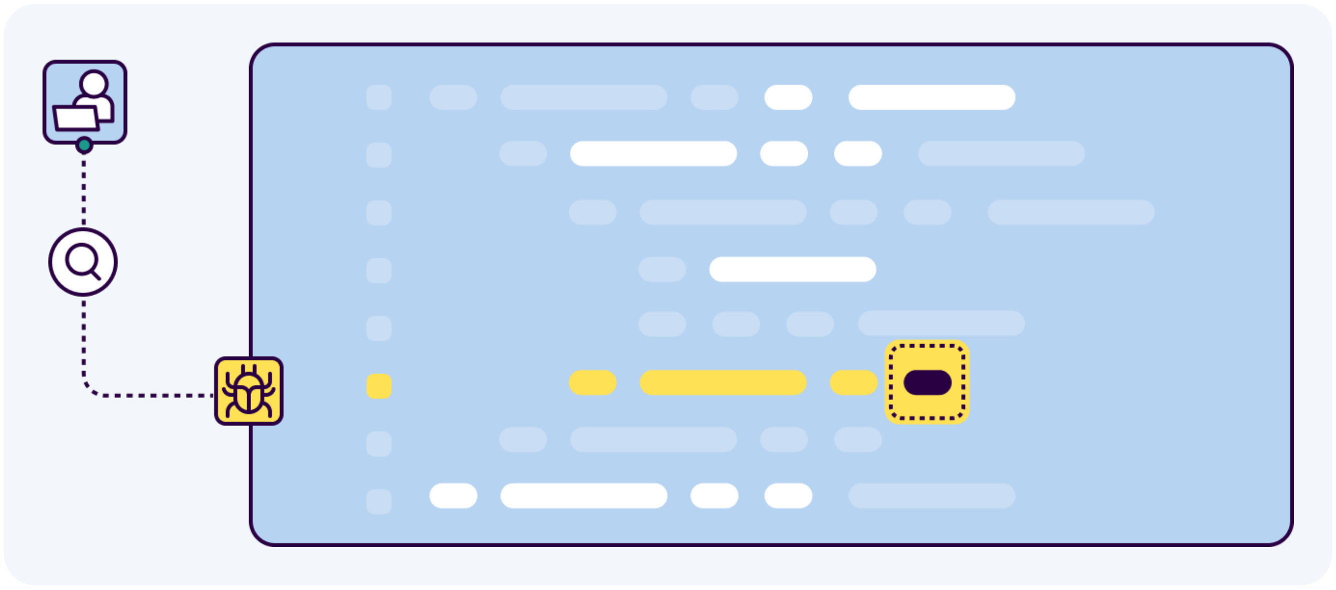
Languages: JS TS **Volume:** □

Mouse and keyboard parity ensures accessibility for all users, including those using assistive technology. This rule flags mouse events that lack corresponding keyboard events. To fix, add the missing keyboard or focus/blur event handlers.

Conditionally executed code should be reachable

Languages: C# Java Python **Volume:** □

Code can become unreachable due to conditional expressions that are always true or false. Developers should review these conditional expressions and either update or remove them.



Why you should fix these common bugs

Statements without side effects

When writing code, it is important to ensure that each statement serves a purpose and contributes to the overall functionality of the program. Statements without side effects or control flow impact typically represent programming errors or redundancy rather than intentional design. Potentially, they can mask actual bugs when developers assume the code works correctly, when in reality critical operations may be incorrectly implemented. At a minimum, these "dead code" statements often indicate incomplete implementations, logical mistakes, or remnants from refactoring that weren't properly removed. This increases performance overhead, maintenance time, and costs as teams investigate code that accomplishes nothing.

Dereferencing null pointers

Even though languages like C# and Java are memory safe, null reference errors are dangerous because they cause unpredictable application crashes and can potentially expose sensitive information through stack traces. These failures often occur only under specific production conditions that weren't caught during testing. What makes these bugs particularly troublesome is that they frequently appear far from their root cause—a null value might originate in one component but only trigger visible failures when used by another component much later in the execution flow. This separation between cause and effect makes null reference bugs notoriously difficult to diagnose, but simple to fix in advance with the right tooling in place to detect them early.

Spotlight: top blocker bug

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 reliability issue found by SonarQube involved unclosed resources in Java, which can cause resource leaks if not properly handled. This bug accounted for over 8% of the Java issues found.

Resources should be closed

Language: Java **Severity: BLOCKER**

Connections, streams, files, and other classes that implement the Closeable interface or its super-interface, AutoCloseable, need to be closed after use to avoid resource leaks. Developers should create the resource using the "try-with-resources" pattern so it will be closed automatically.

Failing to close resources that implement the “Closeable” or “AutoCloseable” interfaces, such as connections, streams, and files, can lead to resource leaks that negatively impact application performance and stability. These leaks occur because the system resources held by these objects are not released when the objects are no longer needed. Over time, this can exhaust available resources, causing the application to slow down, become unresponsive or even crash.

To prevent resource leaks, it is crucial to ensure that these resources are properly closed after they are used. The recommended approach is to utilize the "try-with-resources" statement. This construct ensures that resources declared within its parentheses are automatically closed at the end of the statement, regardless of whether the code block completes normally or throws an exception. It simplifies resource management by eliminating the need for explicit “finally” blocks to close resources, which makes the code cleaner, more readable, and less prone to errors.

Conclusion

This report is intended to be a first step towards increased transparency around some of the most common 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 will help to highlight frequently occurring potential failure points and help developers strengthen the value of the code they create.

About Sonar

Sonar helps developers accelerate productivity, improves code security and code quality, and supports organizations in meeting compliance requirements while embracing AI technologies. The SonarQube platform, used by 7M+ developers worldwide, analyzes all code – developer-written, AI-generated, and third-party open source code – supercharging developers to build better applications, faster.

Sonar provides code review and assurance, inherently applies secure-by-design principles, fixes issues in code before they become a problem, and enforces policy standards – all while improving the developer experience. Sonar is trusted by the world's most innovative companies and is considered the industry standard for integrated code quality and code security. Today, Sonar is used by 400K organizations, including the DoD, Microsoft, NASA, Mastercard, Siemens, and T-Mobile.

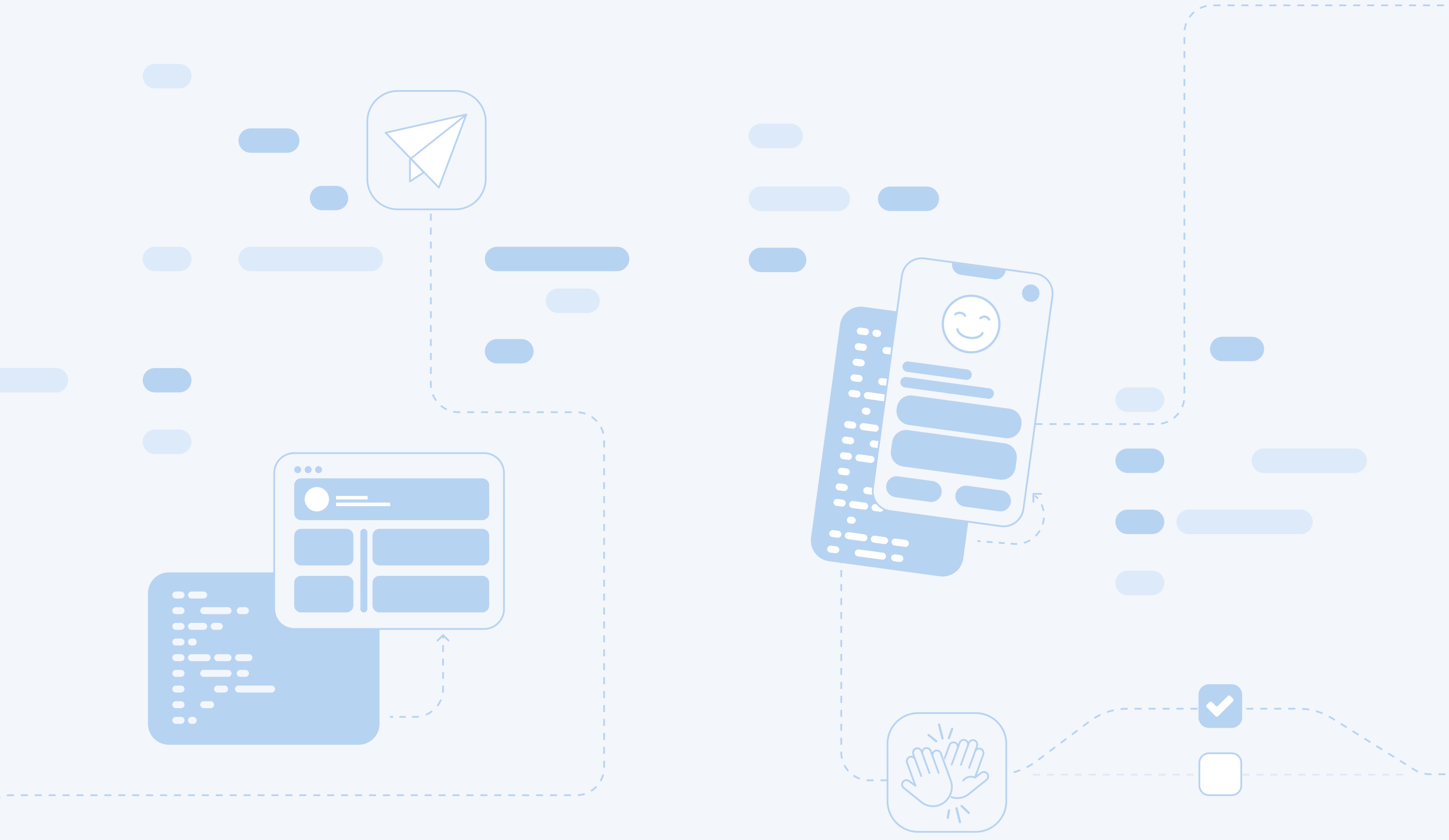
Trusted by over 7M developers and 400K organizations

Microsoft NASA NVIDIA Pfizer Mercedes-Benz

AIRFRANCE BARCLAYS COSTCO WHOLESALE Johnson&Johnson Santander

Learn more at sonar.com

Sonar



sonar.com