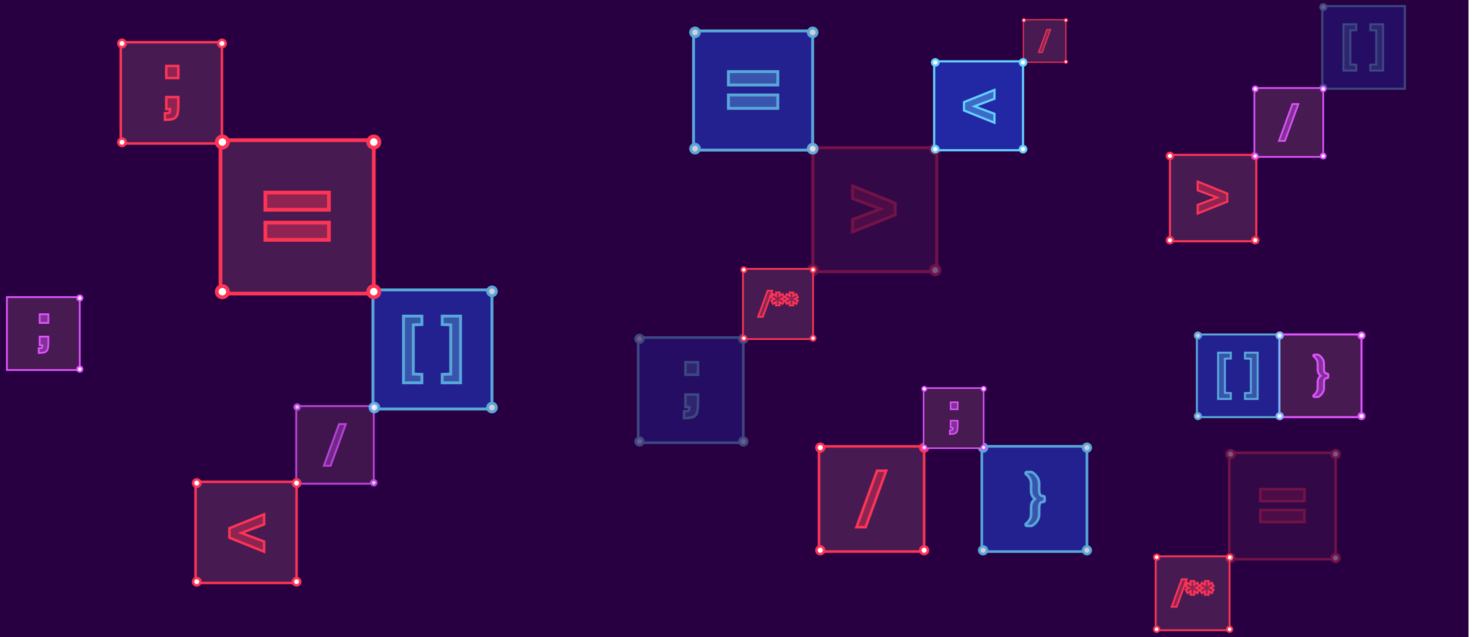
sonar

2023

the cost attributed to code-level technical debt

and how Clean as You Code avoids it



executive summary

Issues in code accumulate over time and can contribute to code-level technical debt. Technical debt leads to lower product quality, increased security risks, reduced developer velocity, efficiency, and morale. Based on an examination of more than 200 projects within a span of 12 months, our research was able to estimate the cost attributed to accumulated code-level technical debt. The study also proposes an alternate way to avoid these costs upfront.

> ***The cost attributable to code-level technical debt over 5 years for a typical project size of 1M Lines of Code (LoC) is estimated at 27,500 developer hours or $1.5M.***

This cost is vast when compounded with an organization's rapidly growing number of projects. Organizations can avoid the cost of bad code with an alternative Clean as You Code approach.

the methodology

Research[1] estimates that developers spend 33% of their time dealing with technical debt, equating to productivity loss and a significant cost to companies annually. Over and above this direct developer cost, the negative impact of technical debt can result in lower product quality, increased security risks, worsening business results, and reduced developer velocity, efficiency, and morale.

Companies choose to tackle technical debt in different ways. One approach is to do nothing or defer action until a later point. This approach, if continued, could eventually require significant refactoring or a complete rewrite of the software. The danger here is that debt accumulates, compounding complexity and potentially exacerbating the impact of code-level issues on the software. As development teams churn, addressing historical issues created by someone else can also lead to greater difficulty, complexity, and frustration.

The methodology proposed by Sonar presents an alternative approach. The Clean as You Code method prevents bad code from reaching production in the first place. It involves focusing on addressing issues in the code that is added or changed so that this code is free from all issues. When all new code is clean, the overall technical debt does not increase, and in fact, progressively reduces over time.

We conducted experiments to estimate the cost attributed to code-level technical debt and provide a quantified directional value of the approach.

This report outlines the process and results.

[1] The Developer Coefficient by Stripe

what is code-level technical debt?

Code-level technical debt refers to the accumulation of unresolved issues during software development. These issues, intentionally or unintentionally left unaddressed, result in future rework and gradually build up over time.

Technical debt generally presents itself differently for every product and company – making it challenging to quantify and compare. It is often a result of excessive software complexity, design flaws, or weak architecture. However, code level technical debt has specific quantifiable characteristics that present opportunities for analysis.

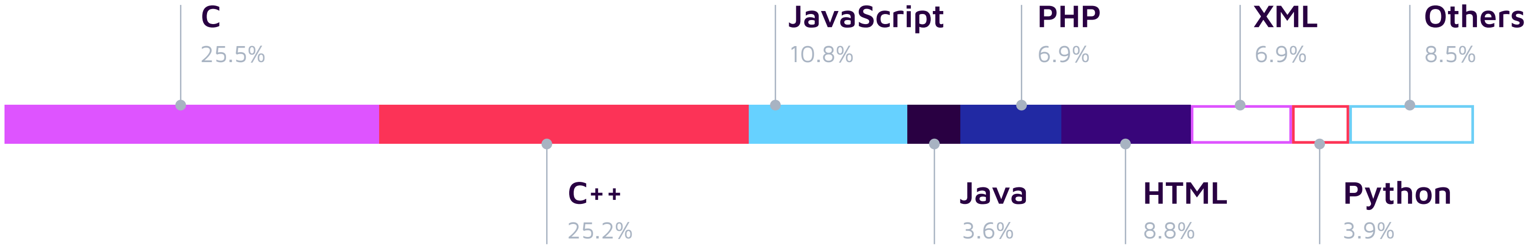
Acknowledging that coding issues are an inevitable part of development, we must recognize that the accumulation of these unresolved issues worsens the impact of code-level technical debt. Addressing these issues becomes more complex and burdensome as time passes, adversely affecting the overall software quality and developer velocity.

our research

We created a quantitative model consisting of a sample set of over 200[2] projects of varying sizes and programming languages to examine the value of employing the Clean as You Code methodology. This distribution enabled us to gather a comprehensive dataset for analyzing the volume and type of issues created over a defined period.

The data extracted totaled approximately 11M Lines of Code (LoC) and covered a variety of programming languages over 12 months. Of the projects analyzed, 27% contained multiple languages, and the average size of projects analyzed was approximately 500K LoC. The chart below illustrates the breakdown by primary language.

SAMPLED PROJECTS CATEGORIZED BY LANGUAGE



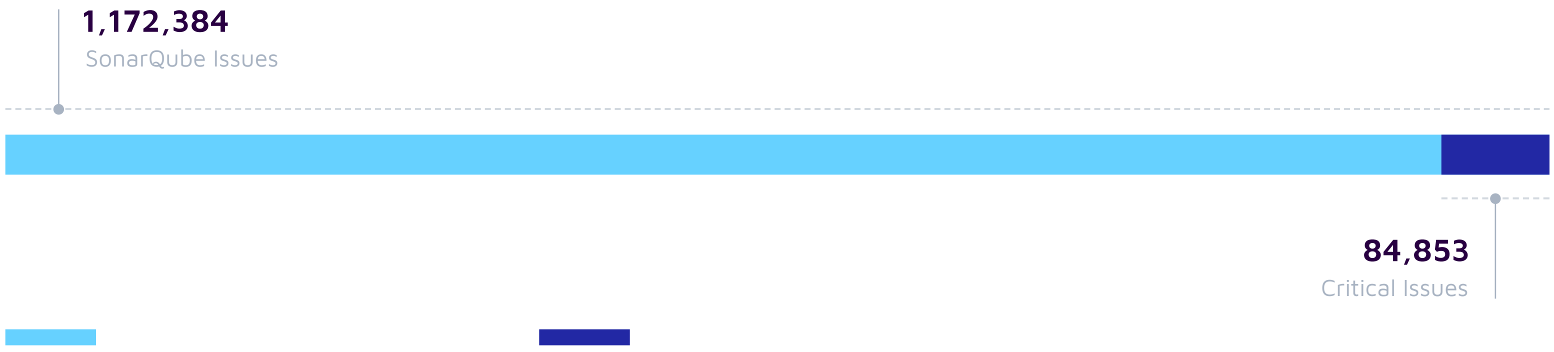
[2] A list of projects analyzed is available here

analysis of newly created issues

The extracted dataset was analyzed using SonarQube (a self-managed code analysis offering from Sonar), which delivered a view of newly created issues per month per project for 12 months.

While it is impractical to describe a typical average project (since it can vary for every organization), the data presented in this report can serve as a guide to estimate the cost associated with accumulated technical debt based on the number of projects and typical project size.

Analysis of the data portrayed a split between issues classified as “critical” and “others.” Critical issues block forward progress and manifest as bugs or vulnerabilities. The category of issues labeled as “others” represents problems in the code that require attention and, if left unattended, may potentially lead to maintainability issues or serious flaws downstream.



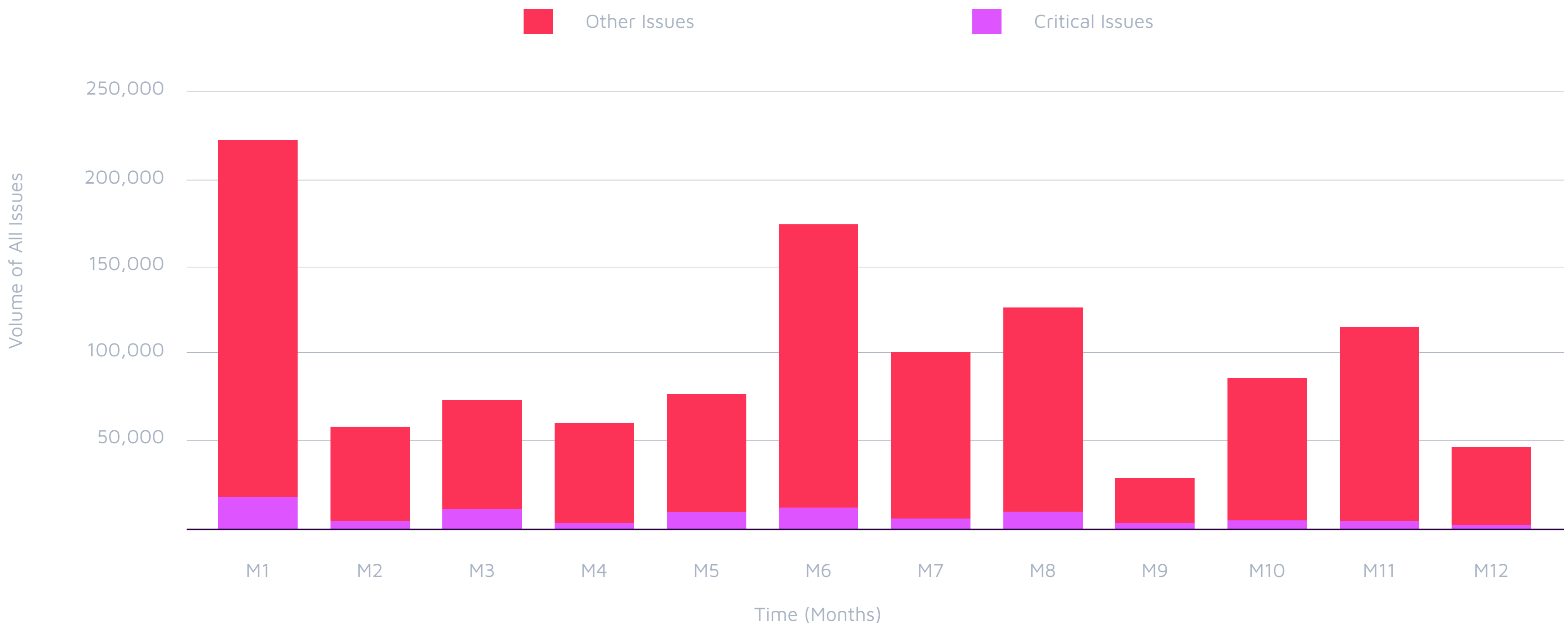
92.8% Other Issues
issues such as code smells that may impact the maintainability and readability of code

7.2% Critical Issues
issues such as bugs and vulnerabilities that can impact code security and quality

Every month, developers contribute to code-level technical debt by creating new issues. The volume of new issues created per month varied over the 12 months across all the analyzed projects. We assume that the project-specific cadence of developer effort and the subsequent merging of new code influence this.

TOTAL VOLUME OF NEW ISSUES CREATED PER MONTH

Across 200+ Projects/11M LoC



estimating the effort required to fix each issue created

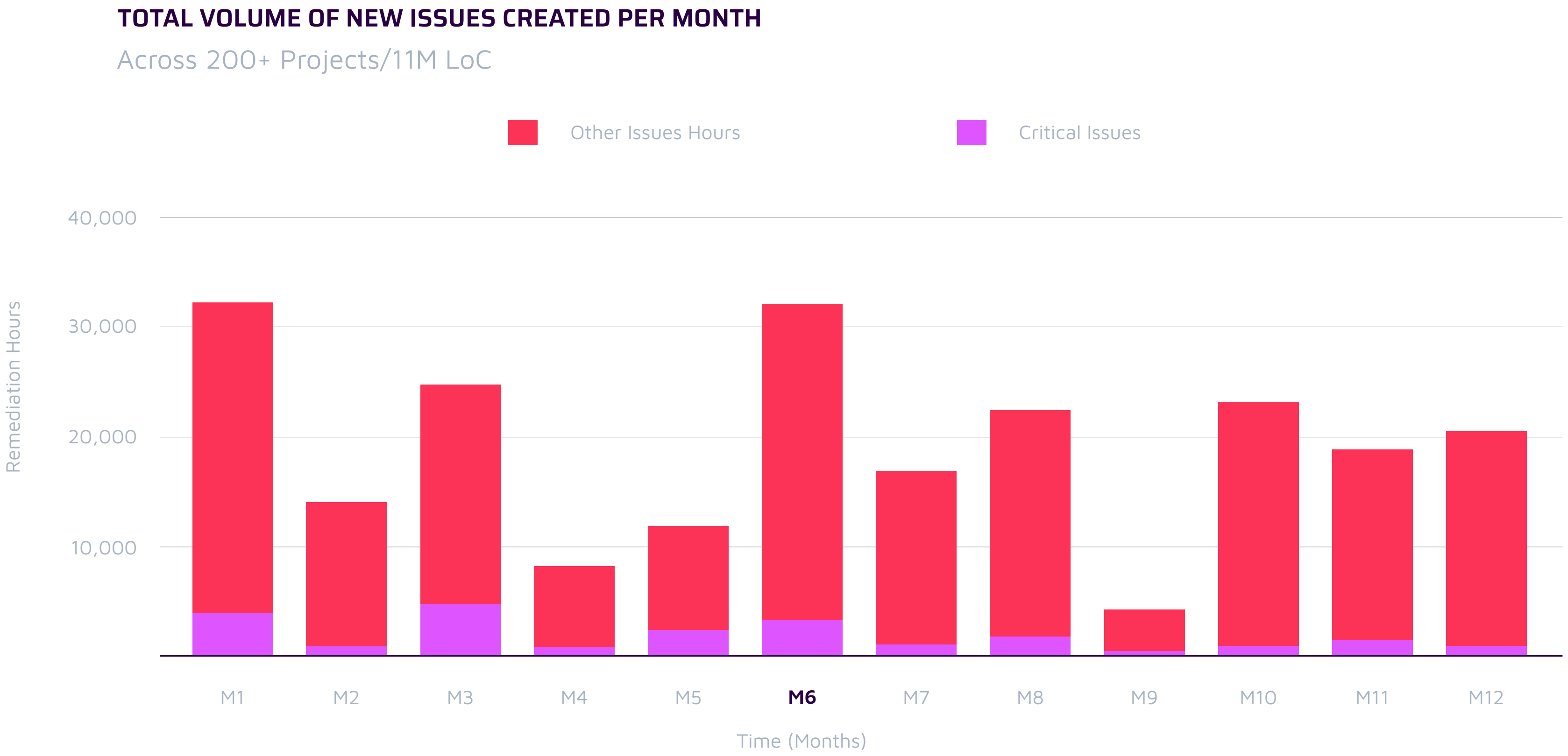
Sonar provides a publicly available library of rules to identify code issues. Each rule has an associated "Effort" or remediation time, calculated through a combination of issue classification and language[3]. Based on this data, we can compute the estimated time to fix the flagged issues. Through this analysis, we computed the approximate time to fix all reported issues to build the total estimated effort (in minutes) required to address the newly introduced issues.

EXAMPLES OF SONAR RULES AND ASSOCIATED EFFORT

Rule	Message	Type	Effort	Severity	Create Date
scala:S1764	Current one of the identical sub-expressions on both sides this operator	BUG	2 min	MAJOR	2022-06-01
csharpsquid:S5542	Use secure mode and padding scheme	VULNERABILITY	20 min	CRITICAL	2022 - 03-05
c:S984	Remove this use of dynamic memory	BUG	1 hr	CRITICAL	2022-03-08
java:S1228	Add a 'package-info.java' file to document the 'test' package	CODE SMELL	20 min	MINOR	2022-01-22
php:S1106	Move this open curly brace to the beginning of next line	CODE SMELL	1 min	MINOR	2022-07-16
java:S2658	Remove this use of dynamic class loading	VULNERABILITY	45 min	CRITICAL	2022-02-17
javascript:S930	This function expects 1 argument, but 2 were provided	BUG	10 min	CRITICAL	2022-08-30
csharpsquid:S3267	Loops should be simplified with "LINQ" expressions	CODE SMELL	5 mins	MINOR	2022-09-25

[3] Adding Coding Rules

calculating the effort to fix newly created issues



Taking **Month 6** as an example—across all the projects surveyed—there were **174k newly created issues**, which would take **32k hours to remediate** or fix.

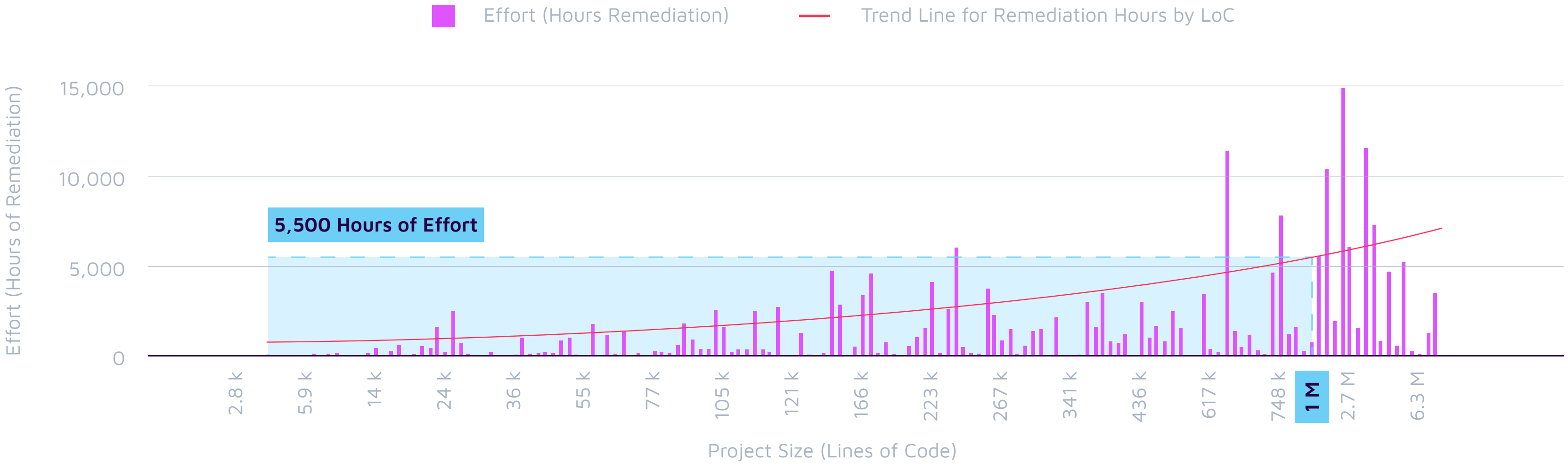
volume of new issues created by the size of repo project

Several factors can impact the volume of new issues created, and we mapped the volume of new issues (and associated remediation time) by the project size based on Lines of Code.

The following plot of “effort” or remediation time versus lines of code shows that the size of the project largely affects the number of new issues created: the larger the project, the more new issues.

REMEDIATION HOURS BY LOC

All Issues



To correlate data across various size projects into relatable insights, we assumed a typical project size of 1M LoC. Based on this assumption, we computed the expected remediation time for all newly created issues over 12 months. The result revealed an estimated 5,500 hours of effort or remediation time required to address all the newly created issues within the 12-month period.

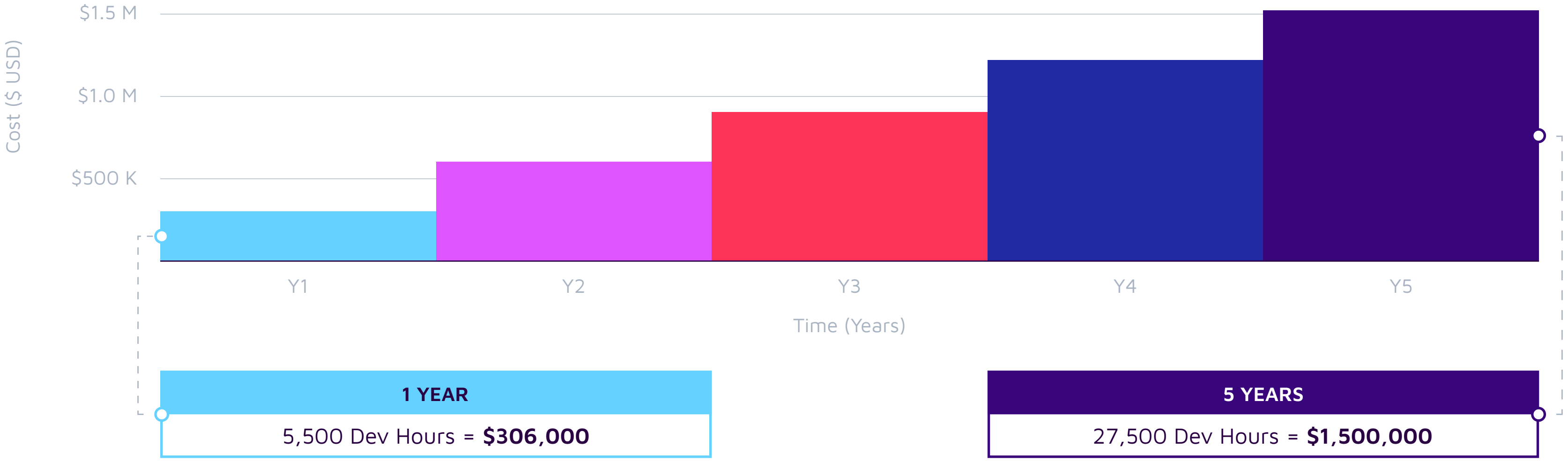
calculating the financial cost

The hours of effort, or developer time, can be converted into an estimated financial cost attributable to the accumulated code-level technical debt.

Assuming the cost of a developer for 1 year (US, fully loaded) at $100,000[4] and the typical 1,800 work hours per year, the cost per hour is $55.56.

5-YEAR CUMULATIVE COST ATTRIBUTABLE TO ALL NEWLY CREATED ISSUES

1M LoC Project



1 YEAR

5,500 Dev Hours = **$306,000**

5 YEARS

27,500 Dev Hours = **$1,500,000**

Assuming that issues are created at the same rate the cumulative attributable cost would amount to **$1.5M over 5 years**.

[4] Stackoverflow survey

the actual impact may be far more significant

In this analysis, we estimated the cost associated with fixing newly introduced coding issues. The cost of every unfixed coding issue progressively increases over time. We estimate that issues left for more than 90 days begin to accrue "interest" as the difficulty in remediating them increases.

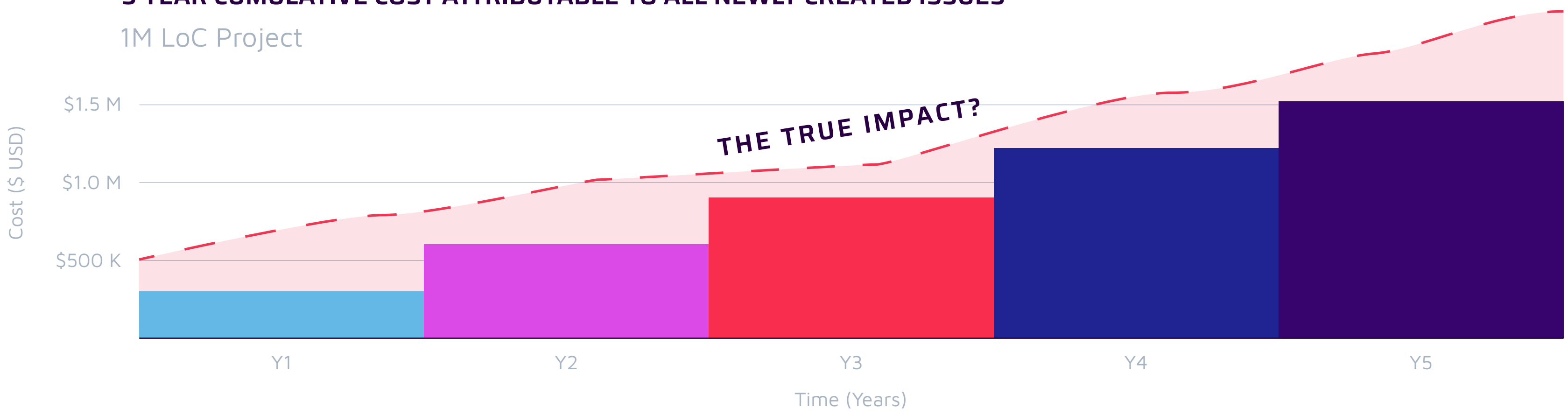
The Clean as You Code methodology identifies and guides developers to fix these issues as they code before they reach production. This proactive approach avoids the increased time and monetary implications that may arise if issues require addressing as part of a future code refactoring effort or are never addressed at all.

Fixing issues as they arise is pragmatic and efficient. Several additional factors contribute to higher after-the-fact remediation costs, or "interest" such as:

- Issues discovered later in production disrupt business
- Longer development time to implement new features due to unmaintainable software
- Loss of business agility because bad code is harder to change
- Developer churn results in getting to grips with errors created by others

5-YEAR CUMULATIVE COST ATTRIBUTABLE TO ALL NEWLY CREATED ISSUES

1M LoC Project



conclusion

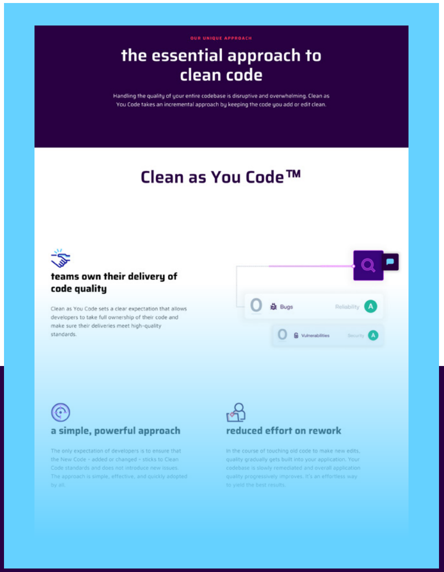
The data extracted from over 200 projects demonstrates that many new issues are created and introduced into code on a regular basis. Unaddressed, these issues accumulate and contribute toward code-level technical debt. The estimated cost attributed to technical debt for a project of 1M LoC was $306K per year or 5,500 developer hours spent on remediation. Over the 5-year lifecycle of an application, these costs could reach $1.5M or the equivalent of 27,500 developer hours.

Given that a developer spends 33% of development time fixing issues in code, the cost to refactor a large portion of an application could be more than 2-3x the cost to fix issues upfront as they occur.

In this study, we focused on attributing the cost of fixing coding issues by examining real-world projects over a defined period and using actual issue remediation times from SonarQube. The actual costs may be far higher when considering the impact issues left unaddressed may have. Debt continues to rise, and addressing issues becomes more complex and burdensome as time passes, impacting the overall software quality. It equates to a significant amount of developer effort and associated costs.

Employing a Clean as You Code methodology allows organizations and their developers to avoid these costs while overcoming the negative long-term impact of technical debt.

resources



OUR UNIQUE APPROACH

the essential approach to clean code

Handling the quality of your entire codebase is disruptive and overwhelming. Clean as You Code takes an incremental approach by keeping the code you add or edit clean.

Clean as You Code™

teams own their delivery of code quality

Clean as You Code sets a clear expectation that allows developers to take full ownership of their code and make sure their deliveries meet high-quality standards.

a simple, powerful approach

The only expectation of developers is to ensure that the New Code - added or changed - sticks to Clean Code standards and does not introduce new issues. The approach is simple, effective, and quickly adopted by all.

reduced effort on rework

In the course of touching old code to make new edits, quality gradually gets built into your application. Your codebase is slowly remediated and overall application quality progressively improves. It's an effortless way to yield the best results.

sonar

Clean as You Code

The essential approach to Clean Code

what is Clean as You Code?

Clean as You Code from Sonar is vital for achieving a Clean Code state - when your codebase has reached a problem-free state and is fit for development and fit for production. This approach enables developers and organizations to optimize the quality of their codebase by solely focusing on code that's added or changed. This simple yet powerful methodology progressively improves the overall quality of the entire codebase with minimal cost and effort.

why is Clean as You Code important?

When teams dedicate less time to addressing old issues or reworking newly created issues, they can accelerate new features, avoid unnecessary rework costs, and foster talent growth and retention. Enabling developers to use Clean Code practices as they code helps achieve these results. Developers can own the quality of their new code while simultaneously improving the existing code they touch in the process. As time passes, old issues get corrected without adding any new ones, all while work on new projects continues with forward momentum. That means teams are delivering what matters most to the business and they get to keep things interesting.

The Clean as You Code methodology can be leveraged across your organization regardless of software maturity, level of developer experience, and internal complexity. By creating and operating on consistent standards with recommended quality gates, the entire development organization can keep its future code clean, regardless of language or platform, project age or size, or existing code complexity. This approach ensures that the future code does not introduce critical security issues, or blocker bugs, and has low technical debt.

what you achieve with Clean as You Code

THE STATE OF CLEAN CODE

Systematically achieve the state of Clean Code, making all code fit for development and fit for production while reducing the burden of technical debt.

DEVELOPER OWNERSHIP OF THE CODEBASE

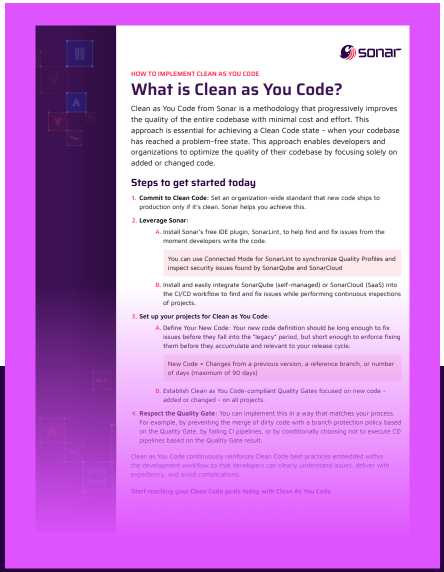
Enable developers to take full ownership of the quality of the code they write with a forward-thinking mentality and leave behind the stress of addressing legacy issues all at once.

ORGANIZATION-WIDE STANDARD FOR CLEAN CODE

The only expectation across the organization is that new code – added or changed – sticks to Clean Code standards and does not introduce new issues. The approach is simple, effective, and quickly adopted by all.

REDUCED EFFORT ON REWORK

Slowly remediate issues and improve quality when touching old code to make new edits. It's an effortless way to yield the best results.



sonar

HOW TO IMPLEMENT CLEAN AS YOU CODE

What is Clean as You Code?

Clean as You Code from Sonar is a methodology that progressively improves the quality of the entire codebase with minimal cost and effort. This approach is essential for achieving a Clean Code state - when your codebase has reached a problem-free state. This approach enables developers and organizations to optimize the quality of their codebase by focusing solely on added or changed code.

Steps to get started today

1. **Commit to Clean Code**: Set an organization-wide standard that new code ships to production only if it's clean. Sonar helps you achieve this.
2. **Leverage Sonar**:
   A. Install Sonar's free IDE plugin, SonarLint, to help find and fix issues from the moment developers write the code.

   You can use Connected Mode for SonarLint to synchronize Quality Profiles and inspect security issues found by SonarQube and SonarCloud

   B. Install and easily integrate SonarQube (self-managed) or SonarCloud (SaaS) into the CI/CD workflow to find and fix issues while performing continuous inspections of projects.
3. **Set up your projects for Clean as You Code**:
   A. Define Your New Code: Your new code definition should be long enough to fix issues before they fall into the "legacy" period, but short enough to enforce fixing them before they accumulate and relevant to your release cycle.

   New Code = Changes from a previous version, a reference branch, or number of days (maximum of 90 days)

   B. Establish Clean as You Code-compliant Quality Gates focused on new code - added or changed - on all projects.
4. **Respect the Quality Gate**: You can implement this in a way that matches your process. For example, by preventing the merge of dirty code with a branch protection policy based on the Quality Gate, by failing CI pipelines, or by conditionally choosing not to execute CD pipelines based on the Quality Gate result.

Clean as You Code continuously reinforces Clean Code best practices embedded within the development workflow so that developers can clearly understand issues, deliver with expediency, and avoid complications.

Start reaching your Clean Code goals today with Clean As You Code.

Clean as You Code Info Hub

LEARN MORE

What is Clean as You Code?

LEARN MORE

Learn How to Implement Clean as You Code

LEARN MORE

www.sonarsource.com

Sonar's industry-leading solution enables developers and organizations to achieve the state of Clean Code. Its open source and commercial solutions – SonarLint, SonarCloud and SonarQube – support 30+ programming languages, frameworks and infrastructure technology. Trusted by more than 400,000 organizations globally, Sonar is considered integral to delivering better software.