Quick Answer: What Are the Best Practices to Improve Code Quality?

Published 19 May 2021 - ID G00745730 - 5 min read

Gunjan Gupta

Initiatives: Software Engineering Strategies; Software Engineering Technologies

Low software quality often constrains future value delivery and reduces team pride in products. Software engineering leaders should improve software quality by investing time and money, and providing their teams with the autonomy to create and manage best practices.

Quick Answer

What are the best practices to improve code quality?

- Adopt a quality-centric leadership approach.
- Give teams autonomy to improve code quality.

More Detail

Gartner inquiry on code quality has increased by more than 15% YoY since May 2019. Figure 1 represents the most common bigrams (two-word associations) found within questions asked around code quality over the last year. The bigrams indicate that there has been much discussion of open source, technical debt and the security of code. More information on methods for secure coding can be found in Integrating Security Into the DevSecOps Toolchain. Practices such as code review and static analysis appear, as do technology solution topics like scanning, automation and detection.
To address these areas properly, software engineering leaders should adopt a quality-centric leadership approach and give teams autonomy to improve code quality.

**Adopt a Quality-Centric Leadership Approach**

High-quality code gives developers pride in their product and gives stakeholders a sustainable application investment. However, many organizations emphasize functional testing and overlook maintainability and other nonfunctional characteristics that are critical to customer experience and team pride. Code that is well architected, documented and maintained, and easily tested, accelerates future value delivery and decreases mean time to repair (MTTR).

To establish code quality as a top priority for their teams, software engineering leaders should:

- Distinguish code quality from developer quality.
- Define measures of code quality — such as code coverage, defect leakage and technical debt — to complement business value metrics.
- Use consistent code quality measures for both internally and externally developed code.
- Promote the idea that teams test early in the development process to increase opportunities for defect detection and remediation.
- Budget for static analysis tool licenses and training.
 Give Teams Autonomy to Improve Code Quality

In addition to using clear metrics and static analysis, software engineering leaders should empower their teams and own the following actions in order to improve code quality:

- Reward teams for delivering business outcomes via teamwork and collaboration. For instance, encourage pair programming or engagement in a community of practice (see Motivate and Engage Learners With Gamification).
- Provide teams with adequate time to install and maintain static analysis and linting tools into their continuous improvement and development environments.
- Challenge teams to write and maintain quality code by allowing them to manage static analysis and linting rule selection. Encourage them to discuss and determine best practices for code architecture and style.
- Ask teams to use pull or merge requests as an opportunity to mentor and to reduce silos through collaborative code reviews.
- Enable quick feedback on code quality by making scans available within the integrated development environment (IDE) and the CI pipeline for net new code (to focus the verbose static analysis findings).
Promote practices such as test-driven development and unit testing to build testable, documented and well-designed code.

Support the collaboration business and operations stakeholders to establish an error budget.

Inventory the licenses of open-source libraries you are using and any code vulnerabilities. To do this, utilize scanning technologies such as software composition analysis (see How to Manage Open-Source Software Risks Using Software Composition Analysis).

Many organizations are very good at functional testing, but functional testing is not enough. Organizations must also address reliability, maintainability and other nonfunctional characteristics. These aspects are critical to customer experience and team pride in their products. Teams that are already adept at practices such as static analysis, unit testing, code reviews or pair programming should focus more on the functional aspects of test automation (see 4 Essential Steps to Implement Test Automation).

Evidence
1 ISO/IEC 5055:2021, ISO.

2 Gartner’s 2020 Achieve Business Agility with Automation, Continuous Quality and DevOps Study was conducted online from June through August 2020 among 205 respondents working for service providers, cloud providers and end-user organizations in North America and Western Europe. Surveyed organizations have deployed or are using DevOps. Qualified organizations had at least $500 million in annual revenue and were required to primarily operate in the banking and financial services, government, insurance, healthcare provider, and retail industries. Respondents were required to work in their organization’s IT function, have a job title less senior than C-Level and be two or more layers away from the most senior executive in their organization.

Respondents’ roles had to be primarily focused on application development, infrastructure and operations or business intelligence and information management. In these focus areas they were also required to perform relevant roles or activities. The study was developed collaboratively by Gartner analysts who follow digital business trends and the Primary Research Team. Results of this study do not represent global findings or the market as a whole, but reflect the sentiments of the respondents and companies surveyed.


Recommended by the Authors
‘Upstream’ Your Software Testing for DevOps

Innovation Insight for Autonomous Testing
Quick Answer: What Are the Essential Shift-Right Practices for Testing in Production?

Optimize Application Testing Quality and Speed With Embedded Intelligent Automation Services

Use Test-First Development Processes to Jump-Start Your SDLC

Improve Software Quality by Building Digital Immunity

Integrating Security Into the DevSecOps Toolchain

4 Essential Steps to Implement Test Automation

Toolkit: PAID Assessment for Technical Debt Prioritization

Address Technical Debt With Gartner's PAID Model and Avoid Bankrupting Your Application's Future

© 2021 Gartner, Inc. and/or its affiliates. All rights reserved. Gartner is a registered trademark of Gartner, Inc. and its affiliates. This publication may not be reproduced or distributed in any form without Gartner's prior written permission. It consists of the opinions of Gartner's research organization, which should not be construed as statements of fact. While the information contained in this publication has been obtained from sources believed to be reliable, Gartner disclaims all warranties as to the accuracy, completeness or adequacy of such information. Although Gartner research may address legal and financial issues, Gartner does not provide legal or investment advice and its research should not be construed or used as such. Your access and use of this publication are governed by Gartner’s Usage Policy. Gartner prides itself on its reputation for independence and objectivity. Its research is produced independently by its research organization without input or influence from any third party. For further information, see "Guiding Principles on Independence and Objectivity."