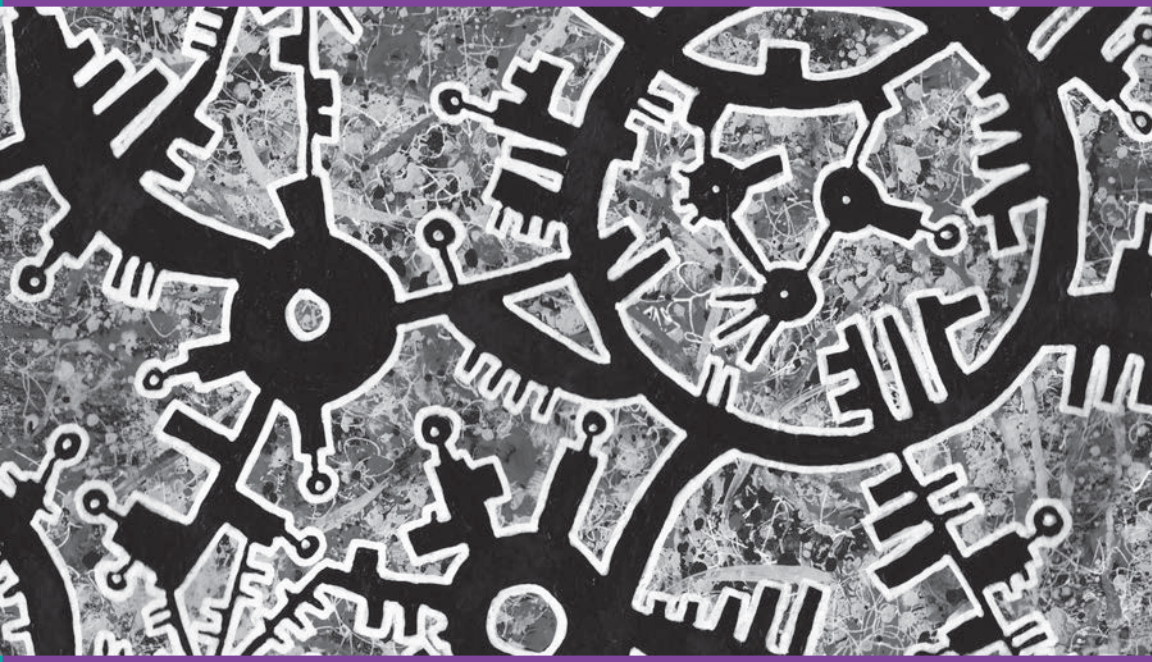
O'REILLY®

Compliments of Software Improvement Group

# Improving Code Quality

## A Survey of Tools, Trends, and Habits Across Software Organizations



Yiannis Kanellopoulos
& Tim Walker

www.sig.eu

# CODE QUALITY CONTROL


BCH operationalizes the 10 guidelines of the book Building Maintainable Software.

**The Better Code Hub** supports dev teams with a Definition of Done for code quality that is crisp, achievable, and fun.

- Actionable, shared **Definition of Done** for Code Quality
- Real-time **status of code quality** for any stakeholder
- **Immediate** developer **feedback** at each commit

Get a free code audit at
**Bettercodehub.com**

bettercodehub.com

# Improving Code Quality

## *A Survey of Tools, Trends, and Habits Across Software Organizations*

*Yiannis Kanellopoulos and Tim Walker*

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

**Improving Code Quality**
by Yiannis Kanellopoulos and Tim Walker



Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://oreilly.com/safari*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

**Editors:** Nan Barber and Brian Foster
**Production Editor:** Shiny Kalapurakkel
**Copyeditor:** Octal Publishing, Inc.
**Proofreader:** Amanda Kersey

**Interior Designer:** David Futato
**Cover Designer:** Karen Montgomery
**Illustrator:** Rebecca Demarest

April 2017: First Edition

**Revision History for the First Edition**
2017-04-10: First Release



978-1-491-98507-6

[LSI]

# Table of Contents

# Preface

How important is code quality, according to practitioners? How do they cultivate and nurture their code quality habits? What obstacles do they experience, and how should organizations facilitate them? SIG and O'Reilly teamed up to find out.

## A Survey of Tools, Trends, and Habits Across Software Organizations

Producing high-quality code is an aim that almost everyone in software development would say they support. Yet, long-term observations in the field reveal that many organizations do not back up that worthy sentiment with the necessary resources (technology, budget, time, training, and management attention) or with the institutional processes required to ensure that the quality of code is routinely maintained.

In 2016, the Software Improvement Group (SIG) collaborated with publisher O'Reilly Media to survey 1,400 software developers on topics related to code quality. The intent of the survey was to uncover trends in overall attitudes, working assumptions, resource distribution, and individual and team behaviors around code quality. In general, the results of the survey reinforced SIG's findings from prior surveys and years of field work with software teams: code quality is valued in principle yet often measured and managed unevenly—or not at all—in the day-to-day practices of software development organizations.

Through their answers to the survey questions, programmers working in a variety of settings convey their experiences on how code

quality is addressed in their organizations. As detailed in this report, survey respondents came from a wide variety of settings, from start-ups to large enterprises, and including both closed source and open source projects.

This report provides detailed results for answers to each question in the survey, along with observations on key correlations among the answers. At the highest level, the survey produced four major findings. Let's take a look at each of them in the sections that follow.

## Responsibility, but Not Enough Facilitation

About *three-quarters* of developers believe that accountability for code quality rests with individual developers and their teams. However, there is a potential disparity between supporting the concept of accountability for code quality and actually having access to the tools and techniques needed to enable a coder or team to ensure an appropriate level of quality. Similarly, almost 80 percent of all respondents said that they address code quality "during coding." Yet more than half of all survey participants use no code quality tools at all.

In other words, even though the developers who participated express a strong commitment to professionalism in terms of code quality, many of them seem to lack the means needed to give a proper foundation to that commitment.

## Lack of Resources

Most developers *do not use tools* for improving software quality. In large part, this is because they lack the budget to acquire them. One part of the problem here was addressed in the previous point: lacking adequate tools, programmers simply will not be able to maintain code quality at the level they would like to. Beyond that, however, lies a deeper issue for organizations, namely that they are not dedicating enough resources—or, in some cases, communicating the availability of those resources—to enable their development teams to deliver high-quality code using a consistent, empirical methodology.

As detailed in "Code Quality Tools" on page 7, more than 70 percent of survey respondents reported that they have no budget reserved for code quality tools—not even a few dollars per month. SIG's hands-on experience with development teams in organizations

across a range of sizes and sectors shows clearly that use of the right tools and methodologies for code quality has a marked impact on the performance, stability, security, and maintainability of enterprise software. In general, paying attention to code quality is the best way to make software "future-proof." Yet, these survey results reveal that most organizations have not embraced this truth, at least as reflected by their budget priorities.

## Inadequate or Inaccessible Tools

Many developers *cannot rely on* typical code quality tools because the tools do not support the relevant technologies and coding languages they use, or else the tools lack certain features that would be of use to them. This might indicate an opportunity for makers of the tools to evolve and improve their offerings to be more relevant for programmers today.

That said, it is worth noting that the technologies most used by survey participants include JavaScript, HTML, CSS, Java, Python, MySQL, and C#, each of which was used by more than 20 percent of respondents. All of these technologies are widely used in software development. Not using code quality tools with these languages might relate less to the features of the tools themselves and more to the lack of budget for them—or simply the lack of awareness of their capabilities among programmers, as addressed in the next point.

## Lack of Awareness or Familiarity

Many developers *are simply unaware* of available tools or are working on teams that have never used them. Institutional inertia might be the main culprit here. Whereas some organizations long ago embraced the use of tools and methodologies that help ensure code quality, many more seem not to understand either the ready availability of tools or the great benefits that come from making code quality an area for rigorous, systematic emphasis.

The following chapters provide detailed assessment of the responses to each survey question. In certain cases, solutions to specific problems will be suggested by referring to other resources provided by SIG.

# Acknowledgements

Special thanks to SIG's CTO, Dr. Joost Visser, and O'Reilly's technical reviewer, Abraham Marín-Pérez, for their invaluable comments on the manuscript of this report.

# Improving Code Quality

## Working Environments of Survey Participants

Survey respondents were asked several questions to determine which technologies and types of projects they work with. Those questions follow in the figure captions, and the results are presented in chart form.

The distribution of developers in Figure 1-1 is probably unsurprising, given the nature of SIG's consulting work and O'Reilly's audience. That said, this survey sample might skew slightly toward developers who work in larger companies. Note, for example, the contrast with figures from StackOverflow's 2016 Developer Survey: although that self-selected survey might contain its own biases, it is worth noting that about 25 percent of StackOverflow's respondents report working in an enterprise of 1,000 employees or more, versus more than 35 percent of the respondents in the SIG/O'Reilly poll who report working in a large enterprise.

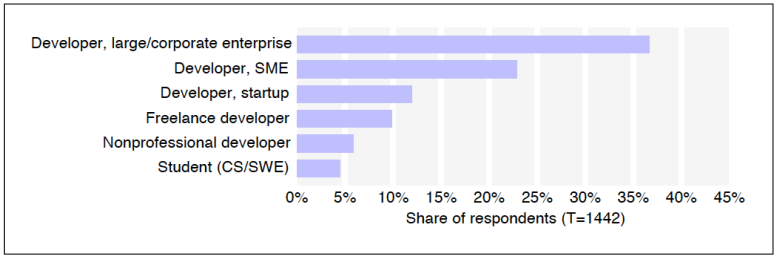



*Figure 1-1. Which option describes your working situation best? (only one answer permitted)*

It might be that large companies stand to benefit the most from giving systematic attention to code quality. When an organization has more developers and a larger code base, implementing best practices for code quality can have a much greater impact on maintainability, security, technical debt, and so on, thereby saving far more resources over time.

Although a clear majority of respondents work on closed source projects (as illustrated in Figure 1-2), more than one-quarter (28 percent) work primarily on open source projects.

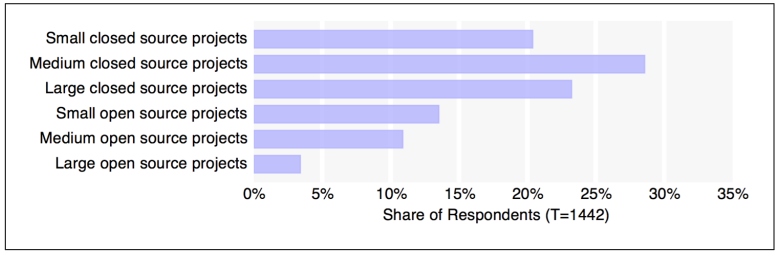



*Figure 1-2. What projects do you typically work on? (only one answer permitted)*

As Rachel Roumeliotis of O'Reilly Media has explained, open source has evolved such that it now includes different types of programmers working from a common ethos: "Open source means everyone contributes—everyone," says Roumeliotis. "Individuals may come from a community background or be affiliated with a large enterprise. Both domains should have a place at the table."

It can be useful, in connection with these results, to consider how code quality affects closed source and open source projects somewhat differently. Closed source projects benefit from improved code quality because they are often created on a custom basis for a specific firm. Achieving higher quality as code is being written therefore means that tomorrow's IT initiatives will benefit from better maintainability, lower technical debt, and more. Meanwhile, by its nature, open source code has the potential to be reused far and wide in ways that might not have been evident to the original programmers—perhaps an even greater impetus to write code that is clean, compact, and modular from the outset.

The technologies most used by respondents to this survey (Figure 1-3) are among the most widely used in the world, as borne out by data from many sources. The high ranking of JavaScript

might also indicate that the survey population leans toward web development more than an average cross section of developers. One topic for potential further inquiry is whether Java and the code quality tools related to it (see "Code Quality Processes" on page 5) are slightly overrepresented in the population of respondents for this survey. It is also possible that there is simply a richer offering of code quality tools for Java than for other languages.

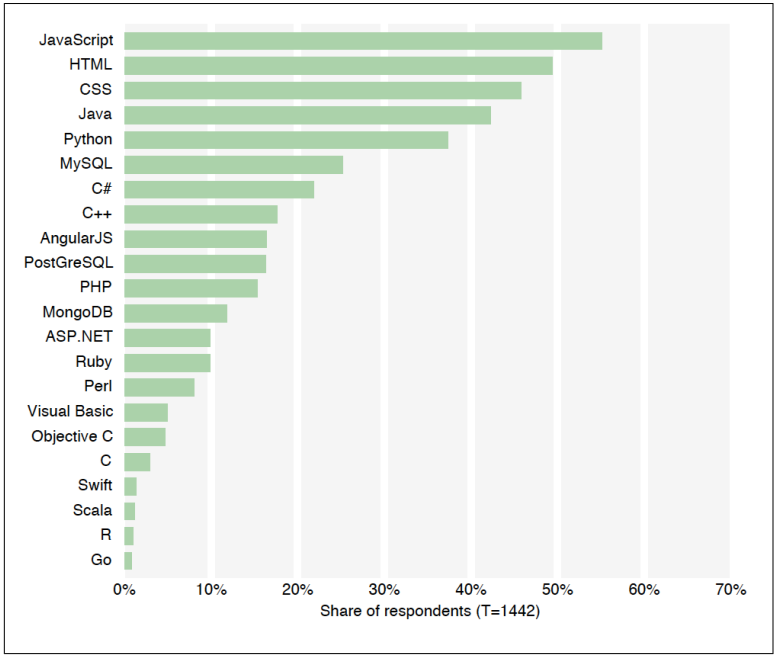



*Figure 1-3. Which technologies are you mostly developing in? (multiple answers permitted)*

Although certain coding elements will always be language-specific, it is worth noting that the 10 principles for writing future-proof code laid out in SIG's report *Building Maintainable Software* (O'Reilly)—"write short units of code," "write simple units of code," "keep unit interfaces small," and so on—can be applied in theory to almost any coding language.

BitBucket is now known as Bitbucket Cloud, and Stash is now called Bitbucket Server. The names in Figure 1-4 reflect the options given at the time the survey was administered.

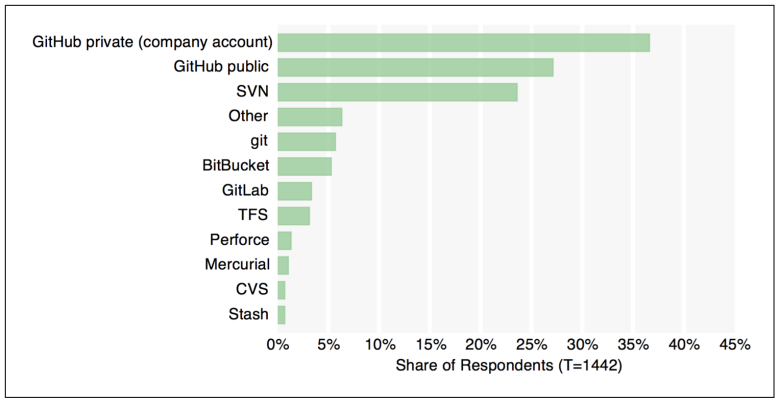



*Figure 1-4. What version control system do you use? (multiple answers permitted)*

GitHub, in its public and private versions, dominates the version control space. (As of February 2017, GitHub claimed to have a community of more than 20 million users, with 53 million projects hosted.) SVN is the only other tool with a share of the sample greater than 6 percent. (GitHub's popularity across many types of organizations and projects is the reason that SIG's online code analysis tools at Better Code Hub integrate with GitHub directly.)

Unsurprisingly, answers to this question correlate with the dichotomy between open source and closed source projects addressed earlier. GitHub public is much more commonly used with open source projects: 48 percent of open source developers use GitHub public, whereas only 19 percent of closed source developers do.

## Accountability for Code Quality

The question in Figure 1-5 aimed at uncovering how survey respondents allot responsibility for code quality within their organizations.

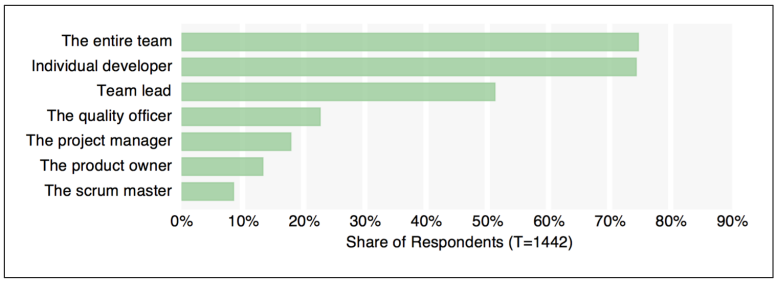



*Figure 1-5. In your opinion, who should be held accountable for code quality? (multiple answers permitted)*

Only 30 percent of respondents chose a single answer; among them, "The entire team" was by far the most common answer, accounting for 20 percent of the entire survey population of 1,442 people.

Conversely, 70 percent of respondents chose more than one response. Most of these respondents—68 percent of the entire survey population, in fact—chose "Individual developer" alongside one or more other options. Almost half of all respondents chose three or more answers to this question, apparently reflecting a belief among most developers that code quality is a broadly shared responsibility.

Sharing that responsibility broadly is not enough; it must also be well defined. As detailed in the SIG's book *Building Software Teams* (O'Reilly), the teams that consistently produce the highest-quality code are those that develop common standards, metrics, and techniques for code quality to create a shared "definition of done" across a software development organization. To reach that point, some teams might need coaching to find the metrics that are important for them. Metrics can evolve with time as team members and requirements evolve.

## Code Quality Processes

The questions in figures 3-1 and 3-2—and the correlations among the answers to them—dig deeper into the specific approaches used to address code quality in survey participants' organizations.

More than three-quarters of respondents (77 percent) selected more than one answer to the question in Figure 1-6. Two answers rose to the top: almost the entire sample (92 percent of 1,442) chose either "During coding," "In code reviews," or both.

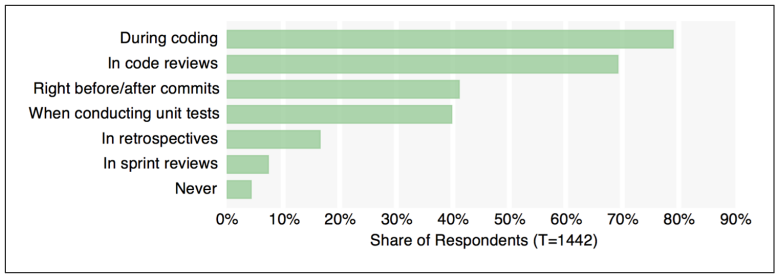



*Figure 1-6. Where in your development process do you address code quality? (multiple answers permitted)*

As might be expected, there were correlations between particular pairs of answers for "In your opinion, who should be held accountable for code quality?" and "Where in your development process do you address code quality?" Developers who placed accountability with the "Scrum master" in the prior question tended to choose "In sprint reviews" for the latter question, whereas those who placed accountability with the "Individual developer" tended to choose "During coding" for the process question.

Thinking ahead to the tool-specific questions in "Code Quality Tools" on page 7, it is worth noting that there was also a correlation between when code quality review takes place and how it is assessed via tools: 76 percent of developers who address code quality during coding are more likely to value support for specific technologies as a feature of a given code quality tool. By contrast, only 48 percent of the developers who do not address code quality during coding cited support for specific technologies as being important.

As shown in Figure 1-7, more than half of the developers surveyed use no code quality tools: 29 percent of respondents answered "None" to this question, whereas 56 percent use peer-led manual code reviews but no other methods or tools.

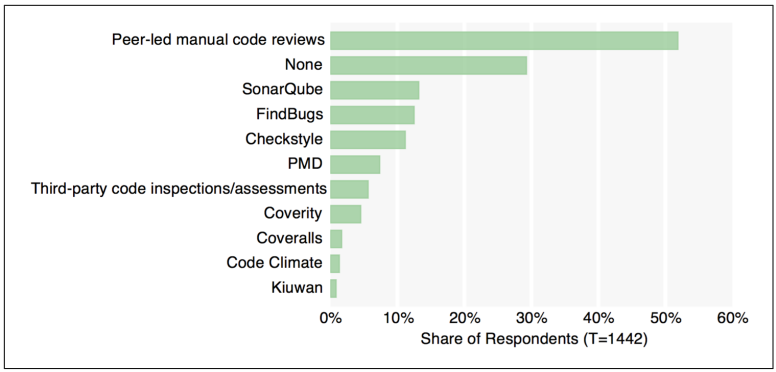



*Figure 1-7. Which code quality methods/tools are you currently using in your projects? (multiple answers permitted)*

About a quarter of all respondents (343 individuals, or 24 percent) used at least one of the four most commonly cited tools—SonarQube, FindBugs, Checkstyle, or PMD. About 5 percent of all developers surveyed used any two of these tools, whereas about 3 percent each used three or four of them. (Use of any of these four tools was also heavily correlated for developers working in Java. See the discussion of technologies in Chapter 1 earlier in this report.)

Use of code quality tools correlates heavily with use of code reviews, even if code reviews do not specifically require them: although only 44 percent of respondents who use no code quality tools reported performing code reviews, fully 80 percent of respondents who use at least one code quality tool participate in code reviews. It might be that a more general awareness of the importance of code quality is reflected on the operational front via code reviews, and as a budget and technology priority via code quality tools.

## Code Quality Tools

The remaining questions in the survey addressed the specifics of code quality tools used (or not) in the organizations of the developers who were polled.

Most respondents to the question posed in Figure 1-8 (71 percent) said that they do not have a budget for code quality tools, and most of the remainder (18 percent) said that they do not know whether they have a budget. That leaves just under 12 percent of the sample who had a code quality tool budget and knew how much it was.

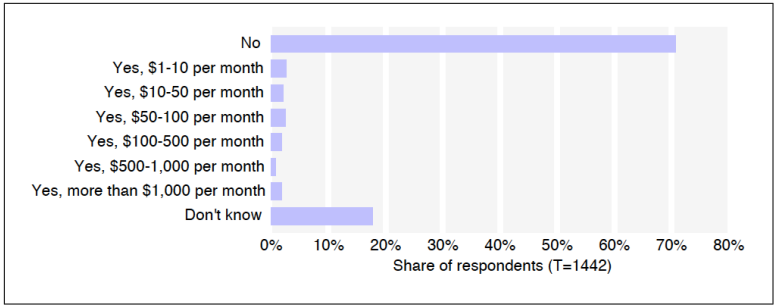



*Figure 1-8. Is there a budget reserved for code quality tools (like Coverity, Checkstyle, SonarQube, etc.) in the projects you work on? (only one answer permitted)*

Not surprisingly, correlating these results with those from the previous question, a number of the code quality tools were more commonly used among those respondents who reported that they had a budget for such tools.

Returning to a point made in the Preface, the lack of a budget for software quality tools inhibits programmers' ability to maintain code quality as they should (and as they say they want to). The lack of a budget also indicates that many organizations, regardless of what they might say about code quality, are not putting their money where their mouths are in terms of allotting the resources needed to ensure that quality code is in fact produced.

The results for the question in Figure 1-9 are not surprising, given the answers to other questions in the survey; for example, the 29 percent of developers polled who answered "None" when asked which code quality tools and methods they use. In this answer, 38 percent reported "Never" using code quality tools, 26 percent reported using them "Sometimes," and 36 percent reported using them "Always" or "Most of the time."

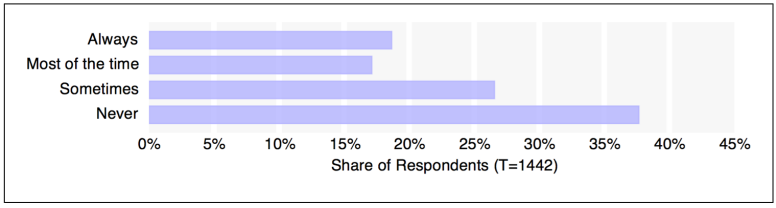



*Figure 1-9. How often do you use code quality tools in the projects you work on? (only one answer permitted)*

Future research might attempt to correlate the frequency of tool use with the size of coding teams (smaller teams might not need as many tools to manage quality in a setting in which peer review would carry more weight) or the expected lifespan of coding projects.

Interestingly, 47 percent of Java developers reported using code quality tools "Always" or "Most of the time," whereas the comparable figure for non-Java developers was only 28 percent. This trend may speak to the ready availability of tools for analyzing Java code, or relate to some other cause not evident from the correlations in this dataset.

The remaining questions in figures 4-3 through 4-9, which address reasons for using (or not using) code quality tools, were asked to subsets of the survey population depending on their answer to the preceding question, "How often do you use code quality tools in the projects you work on?" The specific subset of respondents polled is explained in the figure caption of each question.

For developers who do use such tools "Sometimes," "Most of the time," or "Always," answers were fairly consistent, with tool features, support for specific technologies, and price being common factors for use.

(Because all of the remaining graphs treat subsets of the survey population, the total number of respondents for a given question is included at the bottom of that graph.)

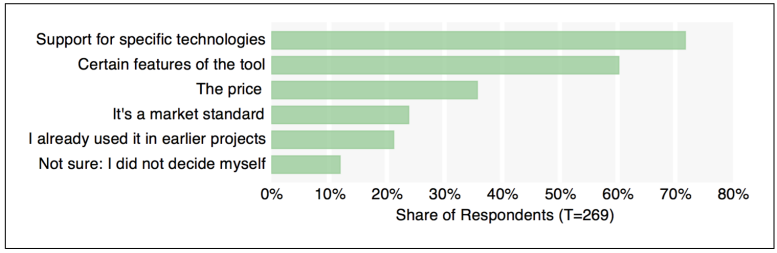



*Figure 1-10. What are the most important reasons to choose a specific code quality tool? (multiple answers permitted; answers from respondents who answered "Always" to "How often do you use code quality tools in the projects you work on?")*

The question in Figure 1-11 (rephrased slightly for different subsets to account for earlier answers) was given to all respondents except

for those who reported "Never" using code quality tools—899 developers in all.

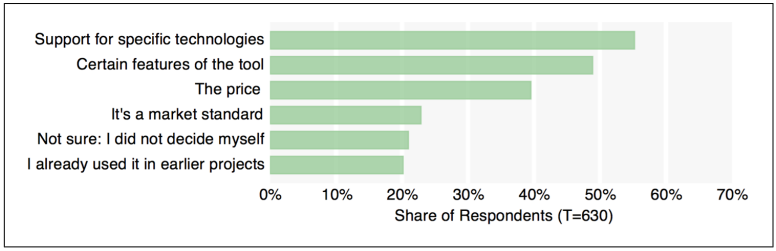



*Figure 1-11. In the cases when you use code quality tools, what are the most important reasons to choose a specific code quality tool? (multiple answers permitted; answers from respondents who answered "Most of the time" or "Sometimes" to "How often do you use code quality tools in the projects you work on?")*

As mentioned earlier, "Support for specific technologies" was an especially popular choice among developers who address code quality during coding. It was chosen by 76 percent of those respondents, compared to 48 percent of those who do not address code quality during coding. It also polled above 70 percent for the developers who "Always" use code quality tools. These results might indicate that organizations can promote more attention to code quality by expending extra effort on finding tools with support and features best suited to regular use during coding. Conversely, software architects and project leaders might want to consider the availability of such tools when they are deciding upon the best language to use for a given project.

It is worth noting that programmers employed by large enterprises are substantially more likely to be unsure why code quality tools are not being used in their projects (Figure 1-12). Specifically, 52 percent of those developers replied that they do not know why such tools are not used, compared to 31 percent of programmers working in all other types of organizations. It might be that programmers in smaller organizations have more of a view into the entire development lifecycle, and are therefore more aware of how decisions are made about the use of code quality tools.

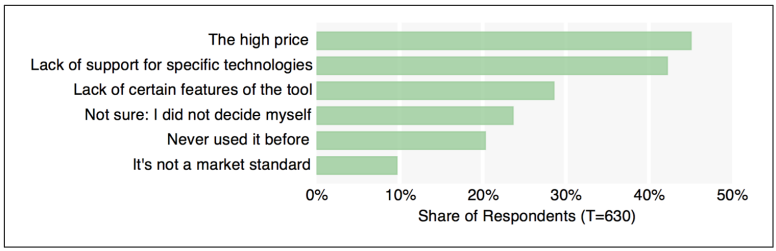



*Figure 1-12. In the cases when you do NOT use code quality tools, what are the most important reasons that influence this decision? (multiple answers permitted; answers from respondents who answered "Most of the time" or "Sometimes" to "How often do you use code quality tools in the projects you work on?")*

Note, also, that money once again becomes a major barrier preventing the use of code quality tools in many cases, even for programmers who often use them. It appears that some developers regularly use tools to ensure code quality when they are working with certain technologies, yet avoid using tools to analyze their work in other technologies because price is seen as prohibitive. If this is indeed the case, it can be worthwhile for an organization to evaluate how this pattern of tool use affects quality, both overall and for the most important technologies in use, and then reallocate money as needed (or even, in some cases, choose different technologies).

As shown in Figure 1-13, it seems that programmers who do use code quality tools find them useful for many different reasons. In response to this question, developers tended to cite multiple answers (5.6 on average), and 13 separate answer choices were selected by at least 20 percent of respondents.

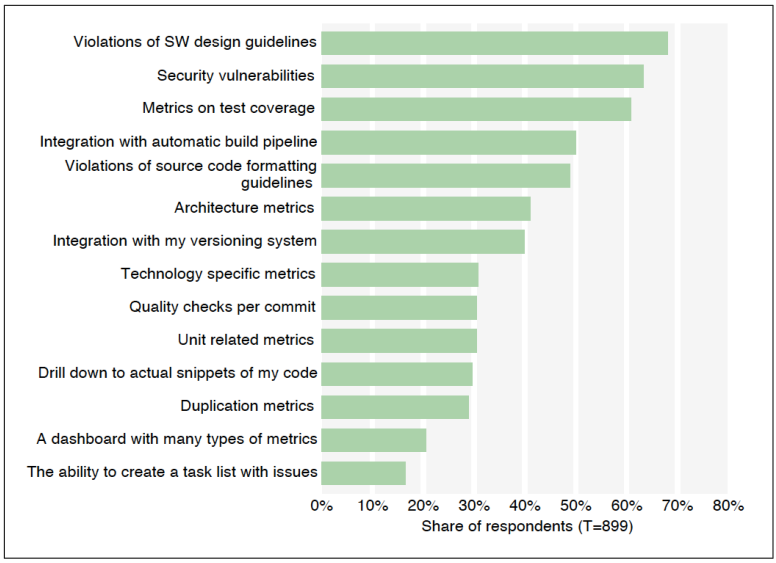



*Figure 1-13. Which features of code quality tools do you consider most useful? (multiple answers permitted; answers from respondents who answered "Most of the time," "Sometimes," or "Always" to "How often do you use code quality tools in the projects you work on?")*

It is important to remember that all of these reasons ought to be related back to the core principles of code quality explained in SIG's *Building Software* and *Building Software Teams*. For example, "Integration with automatic build pipeline" features, if deployed correctly, might support the advice in the "Automate Deployment" chapter of *Building Software Teams*.

Note, also, that care must be exercised when applying the metrics produced by the tools, because they might not be exactly the metrics needed to ensure quality. For more, see *Building Software Teams*' Chapter 2, "Derive Metrics from Your Measurement Goals."

Based on the answers to the question posed in Figure 1-14, most respondents who employ code quality tools fix the majority of the issues that the tools find. Note, however, that a significant portion of respondents reported weak results on this score: 26 percent of these developers said that fewer than 40 percent of the issues found by code quality tools are subsequently fixed.

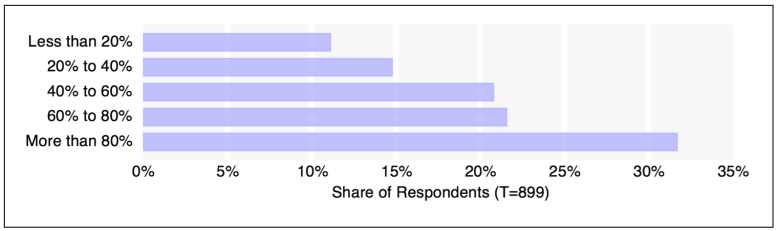



*Figure 1-14. What percentage of the issues that these tools find do you typically fix? (only one answer permitted; answers from respondents who answered "Most of the time," "Sometimes," or "Always" to "How often do you use code quality tools in the projects you work on?")*

This might relate to an issue raised in "Code Quality Processes" on page 5, namely the correlation between performing code reviews and using code tools. Developers and teams that do both might have more rigorous practices when it comes to increasing code quality, so they can pay more attention to the issues found by the tools and have more ways of addressing them. It is also possible that developers take action on fewer alerted issues when they have not had input into the selection or configuration of the tool, or adequate training in its use.

Note, also, that ignoring issues found by tools might relate to the findings of the following question—if, for instance, excessive false positives generated by a tool lead programmers to ignore many of the issues raised.

As illustrated in Figure 1-15, nearly half (46 percent) of developers who use code quality tools cited the excessive generation of false positives as a key problem with the tools, and 36 percent of respondents cited an excessive number of warnings overall. These problems, plus the lack of actionable recommendations cited by about one-quarter of respondents, could help to explain why a significant portion of the developers who use these tools do not fix many of the issues that the tools uncover. (See the previous question in Figure 1-14.)

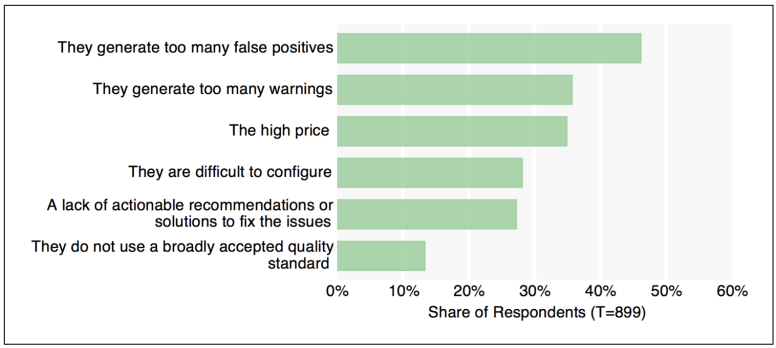



*Figure 1-15. What are the biggest pitfalls of these tools? (multiple answers permitted; answers from respondents who answered "Most of the time," "Sometimes," or "Always" to "How often do you use code quality tools in the projects you work on?")*

Given these findings, some tool makers seem to have an opportunity to refine their products to minimize false positives and perhaps give a higher priority to the most important issues found. There might also be opportunities to better train teams to tune their code analysis tools so that they report the issues that the team really cares about, while omitting others. Meanwhile, the findings support the idea that organizations should invest the time needed to ensure that they are choosing tools that reinforce the core principles of code quality without wasting the time and attention of developers and teams.

When looking at Figure 1-16, it is interesting to note how few respondents gave specific technical or business reasons for "Never" using code quality tools: only 15 percent cited a financial reason, just 12 percent noted "Lack of support for specific technologies," and less than half that number cited market standards. Meanwhile, 63 percent of these 543 respondents cited "Never used it before," "Not sure: I did not decide myself," or both.

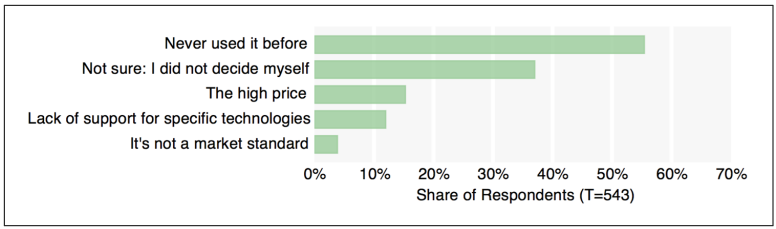



*Figure 1-16. What are the most important reasons that you do not use any code quality tools in your projects? (multiple answers permitted; answers from respondents who answered "Never" to "How often do you use code quality tools in the projects you work on?")*

These results might indicate that failure to use code quality tools arises more from lack of familiarity with them, or lack of understanding about the value of them, than from any specific technical or business reason. As suggested in the Preface, institutional and personal inertia might be the overriding factor. It seems likely that better training for developers, along with allotment of more budget, could have a meaningful impact on the adoption of these tools—with corresponding positive impacts on code quality.

## Further Analysis

There is good news to be found in the results of the SIG/O'Reilly survey. For example, it is a good sign that three-quarters of developers polled consider code quality to be a shared responsibility of the entire team and the individual developer. In other words, most programmers are perfectly willing to be held accountable for the quality of the code they create.

Unfortunately, the results of this poll—combined with decades of SIG field experience—also reinforce the conclusion that too many organizations treat code quality as an afterthought. It is considered an issue that developers must solve for themselves, usually through ill-defined means, rather than a priority that is shared throughout the organization and implemented from the beginning to the end of each software project.

The findings of this collaborative poll with O'Reilly also tend to reinforce earlier research carried out by SIG. That research determined that software development organizations fail to use code quality standards for three main reasons:

- There is insufficient institutional urgency to adopt code quality standards.
- They have not reached internal consensus about what software quality is and how it should be measured.
- They lack management support and a budget to establish and maintain adequate standards.

Overall, software development organizations need better, more holistic approaches to code quality. For example, both individual developers and teams must be trained to have a clear understanding of the dimensions of code quality, including security, maintainability, creation or elimination of technical debt, and developer productivity. Also, code quality initiatives should fully incorporate the needs and viewpoints of all stakeholders—technical, operational, and financial—throughout a project's lifecycle. Only by making code quality an integral part of its ethos and operating practices will an organization create the most value not only with its tools, but, more important, with the coders who use them.

For more than 15 years, SIG has helped businesses and government agencies understand and improve the quality of their software. If you would like to know more about our code quality methodology, please follow this link.

## About the Authors

**Yiannis Kanellopoulos** is the practice leader for Greece at the Software Improvement Group (SIG). He holds a Ph.D. in computer science from the University of Manchester and specializes in helping international organizations manage risks and costs related to the procurement, development, and maintenance of their software systems. Kanellopoulos is also a founding member of Orange Grove Patras, a business incubator sponsored by the Dutch Embassy in Greece to promote entrepreneurship and counter youth unemployment.

**Tim Walker** is a technology writer with decades of experience in journalism, research, and editing. As a journalist, he has published scores of features, reviews, and other articles in venues including the *Austin Chronicle*, *MAKE*, and the *San Antonio Business Journal*. He also served for many years as a high-tech industry editor at Hoover's, and in marketing roles for hardware, SaaS, and data-science startups in his hometown of Austin, Texas.