The 2019 State of Code Review:

Trends and Insights into Collaborative Software Development

SMARTBEAR

SMARTBEAR

The 2019 State of Code Review Report is Presented by SmartBear

We provide high-impact, easy-to-use tools for software teams to plan, build, test, and monitor great software, faster.

SoapUI Supported by SMARTBEAR

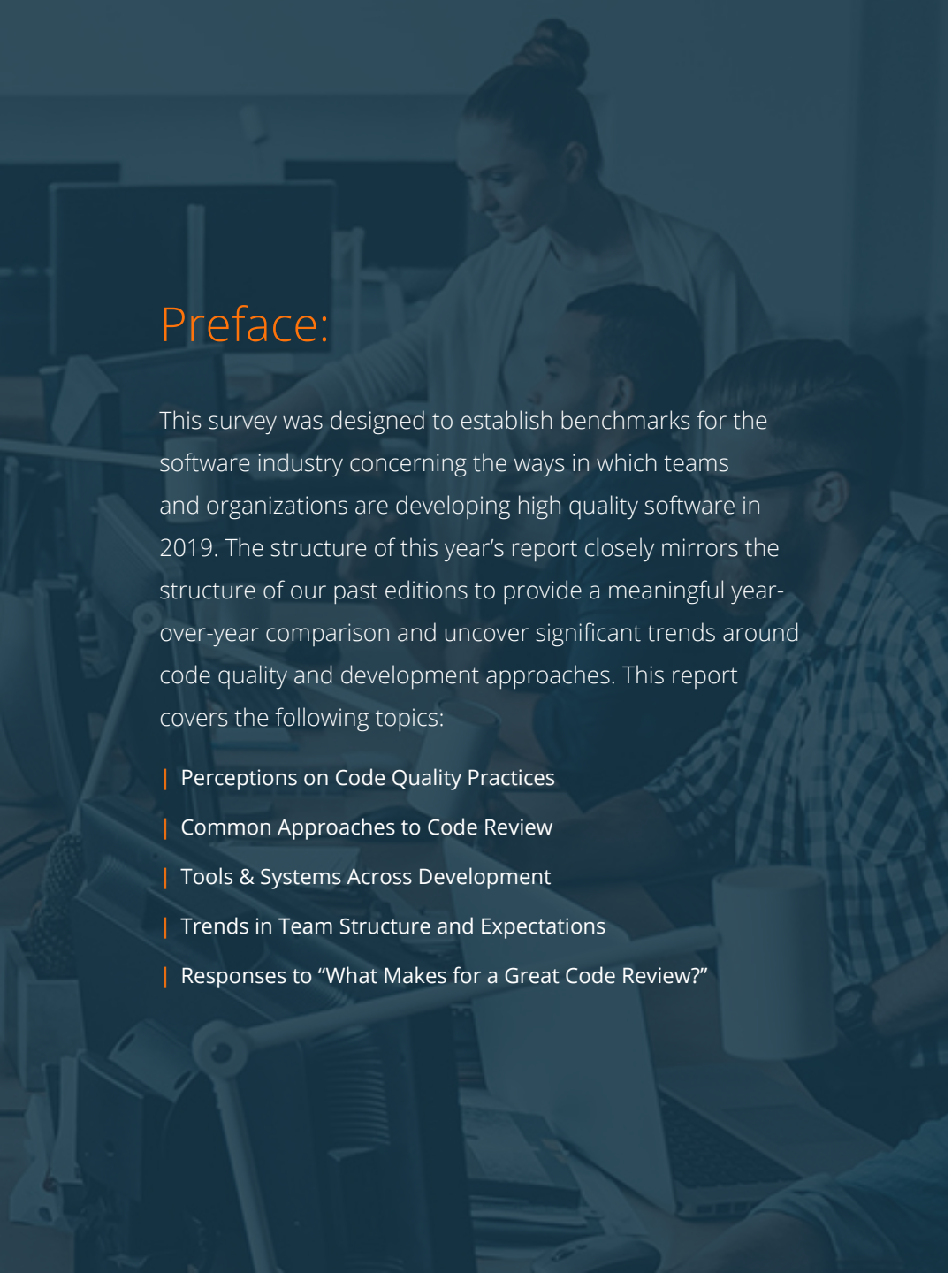
Swagger Supported by SMARTBEAR

SMARTBEAR Collaborator

SMARTBEAR LoadUI Pro

SMARTBEAR ServiceV Pro

SMARTBEAR SoapUI Pro



Preface:

This survey was designed to establish benchmarks for the software industry concerning the ways in which teams and organizations are developing high quality software in 2019. The structure of this year's report closely mirrors the structure of our past editions to provide a meaningful year-over-year comparison and uncover significant trends around code quality and development approaches. This report covers the following topics:

- Perceptions on Code Quality Practices
- Common Approaches to Code Review
- Tools & Systems Across Development
- Trends in Team Structure and Expectations
- Responses to "What Makes for a Great Code Review?"

Methodology:

SmartBear Software conducted a global online survey over the course of ten weeks, from March to May 2019. The findings are based on aggregated responses from more than 1,100 software developers, testers, IT/operations professionals, and business leaders across 35 different industries. Participants in the survey work at companies of all sizes, from fewer than 25 employees to over 10,000. Similarly, their software teams range in size from less than 5 to more than 50.

Contents

Introduction

Over the course of this report, we'll cover a number of subtopics around code review and code quality. To start, these are some of the most high-level software development findings from this year's survey.

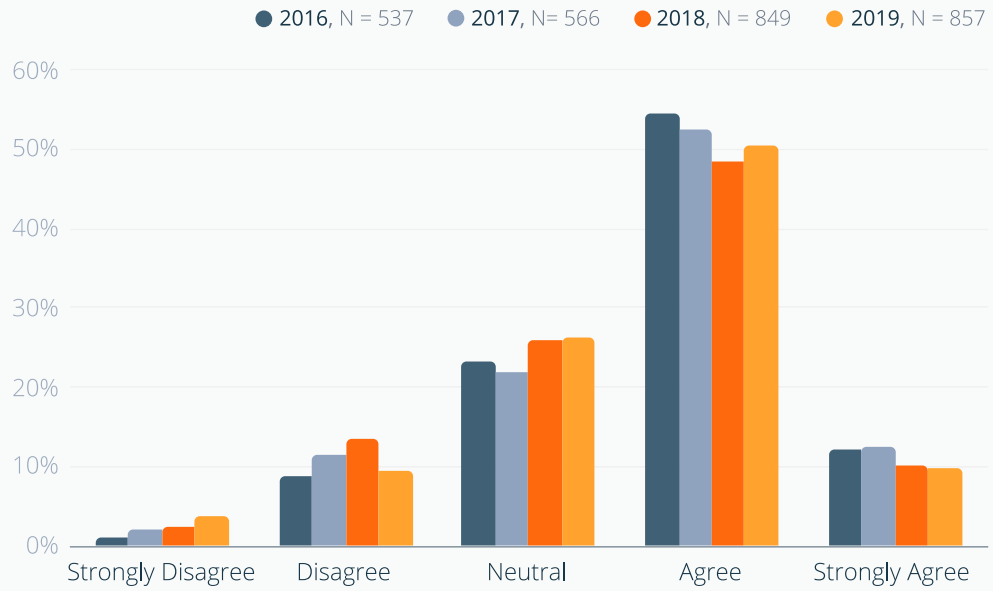
2 out of 3 of respondents are satisfied with their code quality.

Last year, we noted a downward trend in satisfaction with software quality. This year, satisfaction has rebounded slightly. This year's data shows that despite small changes each year, respondents tend to have a pretty positive view of the quality of their team's work *(fig.1)*.

In previous years, higher satisfaction with software quality corresponded with meeting deadlines. But while software-quality satisfaction rebounded this year, release accuracy followed a different trajectory.

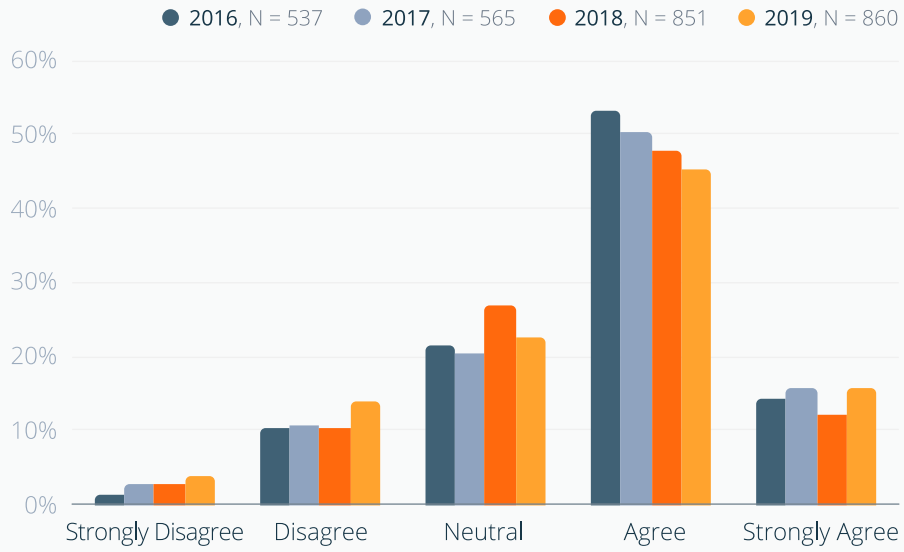
I am satisfied with the overall quality of the software I help deliver.

fig.1



My company is regularly able to get releases out on time.

fig.2



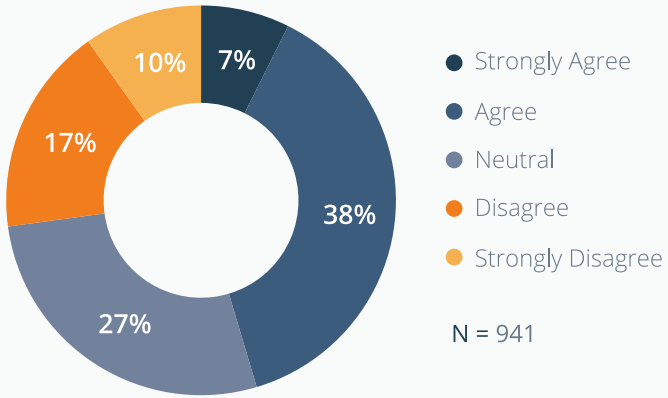
There seems to be polarized reactions when it comes meeting release dates. Of the teams that are able to get releases out on time, more of them selected "strongly agree" than any other year surveyed, while "agree" continued to decline *(fig.2)*.

In 2017 and 2018, ~13% of respondents disagreed or strongly disagreed about getting releases out on time. This year, that figure is up by 5 points to 18%. More teams have figured it out, but more teams are struggling.

It's worth noting that this chart doesn't speak to the frequency of releases so much as broad deadline success. Section #4 of this report includes new data on how often teams are releasing and the challenges that prevent them from meeting their deadlines.

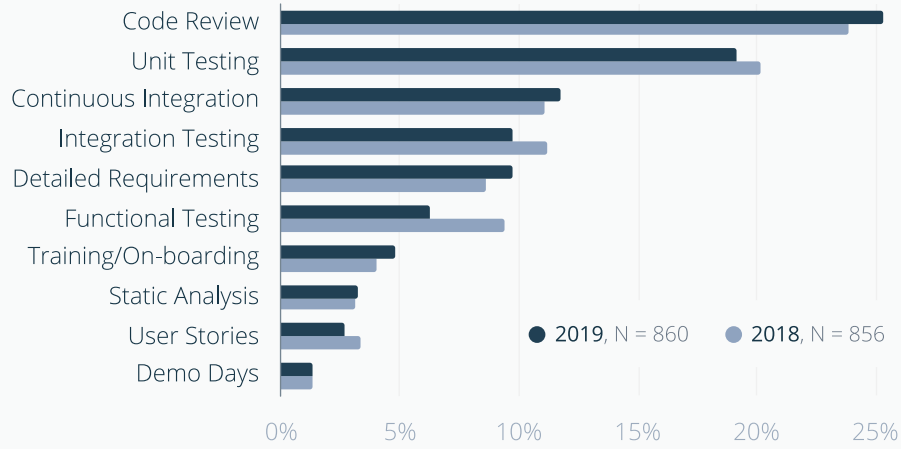
I am satisfied with my team's current code review process.

fig.3



What is the #1 thing a company can do to improve code quality?

fig.4



Only 45% of respondents are satisfied or very satisfied with their current code review process.

Last year, we added a question on code review satisfaction to our survey *(fig.3)*.

Since code reviews can seem like added work, it's reasonable that many developers would be indifferent or neutral to their process satisfaction. For many respondents, they might not have much control over their own process, or it might be less formally defined.

More respondents are satisfied to some degree with their code review process (45%) than dissatisfied (27%). Later in this report, we will reference these two cohorts to better understand how satisfaction with one's code review process relates to other development practices.

For the 4th straight year, respondents identified code review as the #1 way to improve code quality.

Since 2016, respondents to our annual report have overwhelmingly identified code review as the top practice for improving code quality *(fig.4)*. Each

year, unit testing has followed as the 2nd highest ranked choice. This year's data shows the same ranking, with 25% of respondents once again selecting code review.

The perception of code review being a key to better software quality is validated by this year's data. We compared overall software satisfaction between two cohorts: "unsatisfied with their code review process," and "satisfied with their code review process."

The chart to the right shows the correlation between code review satisfaction and satisfaction in overall software quality. 80% of respondents who were satisfied with their code review process were also satisfied with the overall quality of their software. Respondents who were not satisfied with their code review process were less than half as likely to be satisfied overall, accounting for only 35% of that cohort *(fig.5)*.

Of all the measures and attributes we surveyed, satisfaction with one's code review process was the most likely indicator of satisfaction with overall software quality. Over the course of this report, we'll show year-over-year changes in code review practices and identify common traits that contribute to a better peer review approach.

Correlation between code review process satisfaction and code quality satisfaction. *fig.5*

Software Quality Satisfaction

Code Review Satisfaction	N = 856	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
	Strongly Agree	0%	8%	11%	44%	37%
	Agree	0%	3%	18%	66%	13%
	Neutral	1%	6%	37%	50%	6%
	Disagree	6%	18%	33%	39%	3%
	Strongly Disagree	22%	29%	26%	20%	2%

Section 1:
The Code Review Process

Whether development teams find their process through natural iterations or intentional requirements, there are countless configurations of workflows, frequencies, toolsets, and behaviors. In this section, we'll look at what teams are doing today.

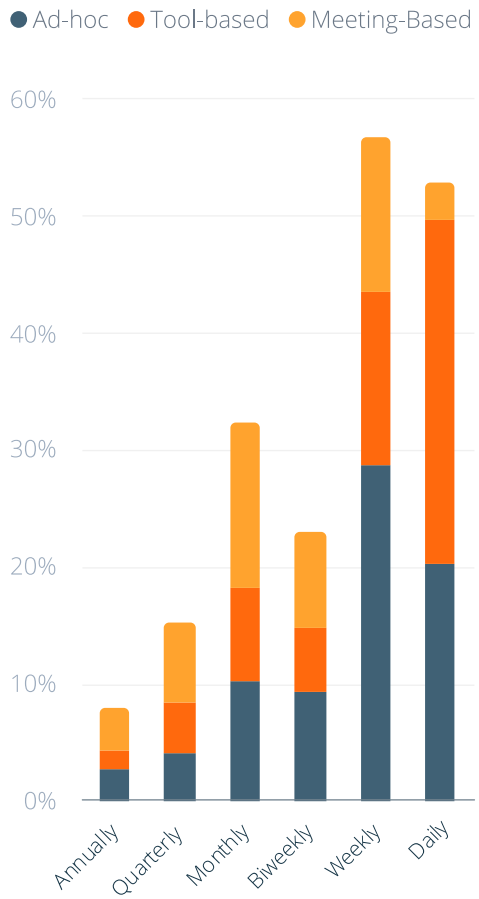
Highlights

- **67% of respondents** are participating in code reviews at least weekly, with 42% reviewing code daily.
- **49% of respondents** are participating in "ad hoc," or over-the-shoulder reviews, at least weekly, with 20% reviewing this way daily.
- **44% of respondents** are participating in tool-based code reviews at least weekly, with 29% reviewing this way daily.
- **16% of respondents** are participating in meeting-based reviews at least weekly, with 3% reviewing this way daily.

These three methods (ad hoc, meeting-based, and tool-based) are the most common approaches to code review. This chart shows the distribution of review type and frequency in this year's survey *(fig.6)*.

How often do you participate in these common code review approaches? *fig.6*

N = 1106



Ad Hoc Reviews

Over the last three years, there's been a steady increase in the frequency and prevalence of ad hoc reviews. These are some of the key metrics in this year's survey that show that change:

- **20%** of respondents are conducting ad hoc reviews on a daily basis. This has been increasing steadily, with that figure at 18% in 2018, 16% in 2017, and 14% in 2016.
- **57%** of respondents are conducting ad hoc reviews on at least a biweekly basis. This number is flat year over year, but had been on the rise with 53% in 2017 and 43% in 2016.
- **22%** of respondents are not conducting ad hoc reviews at all. This is flat year over year, but had been down respectively from 25% and 28% in the two years prior.

Tool-Based Reviews

From 2016 to 2018, there hadn't been much change to the frequency and prevalence of tool-based code reviews. The only significant change was a 6% increase from 2017 to 2018 in respondents conducting some form of tool-based reviews. In our 2019 survey, we found that teams are performing daily tool-based reviews more than ever before.

- **29%** of respondents are conducting tool-based reviews on a daily basis. This is a dramatic increase. That figure has been relatively flat in years prior: 21% in 2018, 21% in 2017, and 23% in 2016.
- **49%** of respondents are conducting tool-based reviews on at least a biweekly basis. This is up over years prior where that figure was 47% in 2018, 48% in 2017, and 44% in 2016.
- **66%** of respondents are conducting tool-based reviews, which is flat year over year, after a slight increase from 60% in 2017 and 63% in 2016.

Meeting-Based Reviews

Over the last few years, we have noted that respondents seemed to be participating in more meeting-based reviews, despite still being in low single digits. We attributed that to new communication platforms like Slack, Teams, Stride, and Flock, which have made it easier than ever for teams to connect. Meetings that once required participants to be in the same room are now easily setup with video chat tools.

Compared to last year's data, meeting-based reviews in 2019 are less common.

- **3%** of respondents are conducting meeting-based reviews on a daily basis. This is down from past years: 5% in 2018, 6% in 2017, and 4% in 2016.
- **30%** of respondents are conducting meeting-based reviews on at least a biweekly basis. After reaching 34% in 2018, this figure is back down to the 30% we saw in 2017.
- **52%** of respondents are conducting meeting-based reviews. This number is down significantly from past years: 65% in 2018, 60% in 2017, 63% in 2016.

Last year, we identified teams conducting meeting-based reviews as an indicator of higher overall code-review satisfaction – this year's data points to the same conclusion. 53% of teams satisfied with their code review process have review meetings, compared to 39% of teams dissatisfied with their process.

Document Reviews

In addition to peer code reviews, we're also tracking how teams are reviewing documents year over year. This year, we found that 86% of respondents participated in some type of document review. This chart shows the types of documents most commonly reviewed, and how it compares to last year's data *(fig.7)*.

There seems to be a year-over-year decline across the board. The only clear reason would be the overall decline in respondents who review any type of document. Last year, 90% of respondents said that they participate in document reviews of some kind. That number declined by 4% this year.

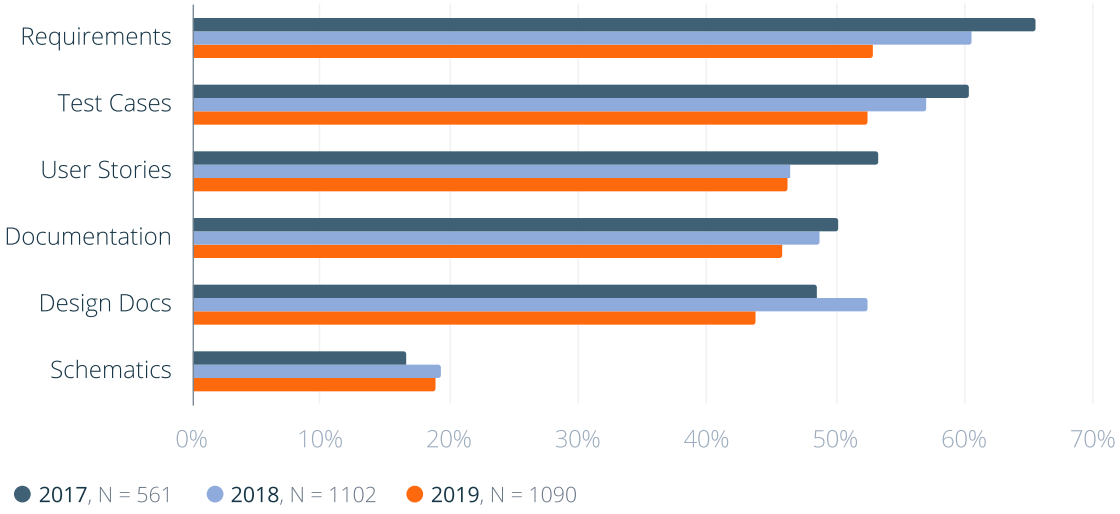
Despite 86% of respondents saying they participate in document reviews, only 34% said they use a tool to review it. This figure has remained flat over the last two years. Most teams are likely conducting document reviews through general communication means, whether in a meeting, chat, or email.

Cohorted by overall code satisfaction, 43% of respondents satisfied with their code review process have a tool for reviewing artifacts. That number was only 26% for respondents in the dissatisfied code review cohort.

Additionally, the satisfied code review cohort responded that they review on average 2.8 types of documents. The dissatisfied cohort responded that they reviewed on average 2.1 types of documents.

This data suggests that teams who conduct more document reviews with a tool are more likely to have a satisfactory code review process and therefore higher code quality satisfaction. One reason could be the learning experience that occurs during reviews of requirements or test-cases, leading to more aligned development.

Which of the following artifacts do you review, if any? *fig.7*



Section 2:
Perceptions on Code Review

We know a satisfactory code review process is critical to producing satisfactory overall code quality, so we asked respondents how they conduct and think about reviews. This section looks at common benefits, use cases, and business drivers around their process.

Highlights:

- **54%** of respondents say that their team has guidelines for how reviews should be performed.
- **59%** of respondents say that their team uses code review to onboard and train new developers.
- **91%** of respondents say that the most important benefit of code review is "Improved Software Quality," followed by "Sharing Knowledge Across the Team" at 82%.

The Top Benefits of Code Review

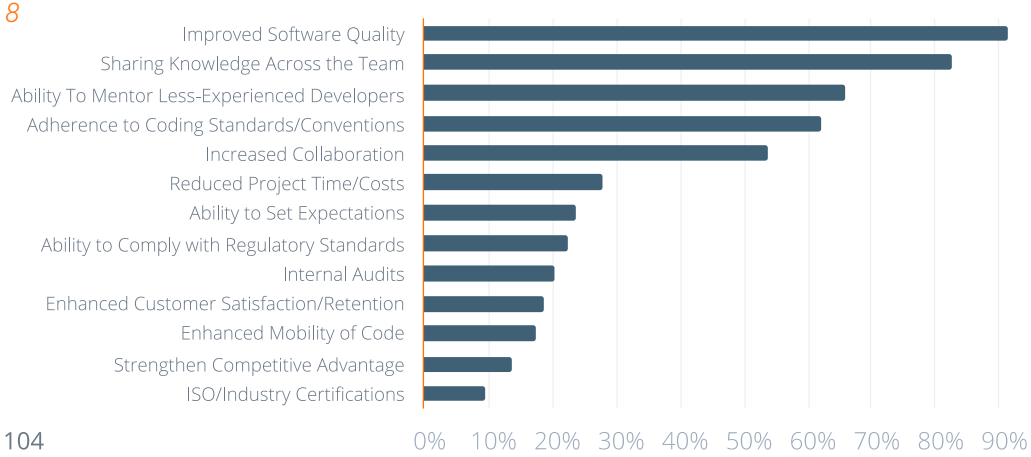
Just as there are a wide variety of code review workflows and approaches, there are also different benefits that teams derive from reviews. Finding bugs and defects is the core function, but the process itself can improve communication, transfer knowledge, and increase standardization.

We asked participants to share what they see as the main benefits *(fig.8)*.

"Improved Software Quality" and "Sharing Knowledge Across the Team" have remained the top two benefits since 2015.

What do you believe are the most important benefits of code review?

fig.8



N = 1104

Compared to the last two years, these are the responses that have changed most:

Benefits of Code Review	2019	2018	2017	Net Change
Gaining Attention				
Ability to Mentor Less-Experienced Developers	**66%**	57%	55%	+11%
Adherence to Coding Standards/Conventions	**62%**	59%	52%	+10%
Sharing Knowledge Across the Team	**82%**	73%	74%	+8%
Losing Attention				
Enhanced Customer Satisfaction/Retention	**19%**	26%	26%	-7%
Reduced Project Time/Costs	**28%**	37%	33%	-5%
Strengthened Competitive Advantage	**14%**	20%	18%	-4%

These changes seem to indicate that teams are increasingly focused on the internal benefits over the external market benefits. Mentorship and knowledge sharing lead to stronger teams and individual growth. More abstract benefits like reduced project time, customer satisfaction, and strengthened competitive advantage don't resonate as strongly.

Business Value of a Review Tool

With these benefits of code review outlined, we also asked about the business drivers that have led teams to adopt a code review tool *(fig.9, next page)*.

This year, "Improving Code Quality" (80%) and "Recent Bugs/Outages" (36%) were the top two drivers for why teams are using a code review tool. With evolving ISO standards and government regulations, some development teams have to prove that they've taken sufficient software assurance measures. Code review offers those teams a documentation event whenever changes are made that could potentially result in new defects. Whether for internal or external audits, recorded peer reviews are a useful way to build a software audit trail.

Across the board, audit and compliance reasons were not as important business drivers as in past years. This could be the result of a smaller percentage of survey respondents doing work that needs to be audited, but we can't say definitively.

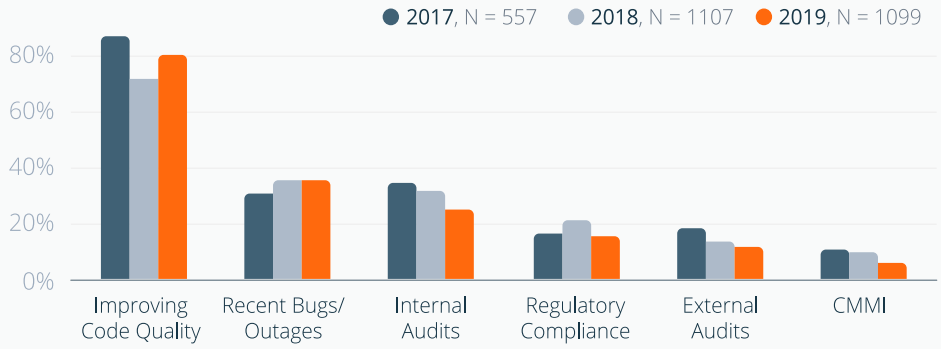
On average, each respondent chose two of the reasons here. 13% of respondents answered "Not Applicable" as an option for teams that don't conduct tool-based reviews, flat compared to 2018.

"Recent Bugs/Outages" has continued to increase year over year, with 31% in 2017, 35% in 2018, and now 36% in 2019. This is understandable since software is becoming more complex and difficult to maintain. The last few years have seen record-setting levels of software-caused recalls across industries like automotive, healthcare, and consumer devices. Additionally, new cybersecurity challenges are forcing teams to address vulnerabilities with more caution and nuance than ever before.

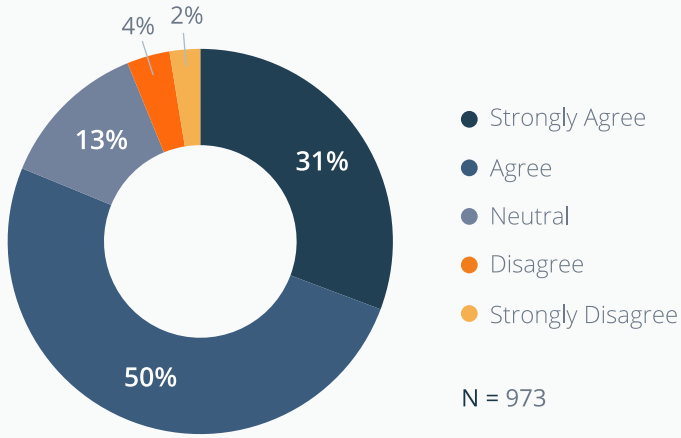
Learning and Onboarding with Code Review

Earlier in this section, we highlighted how respondents are increasingly seeing their code review practice as a way to "Share Knowledge Across the Team" and "Mentor Less-Experienced Developers." To get more explicit, we introduced new questions this year to verify that respondents are frequently learning during code reviews *(fig.10)*.

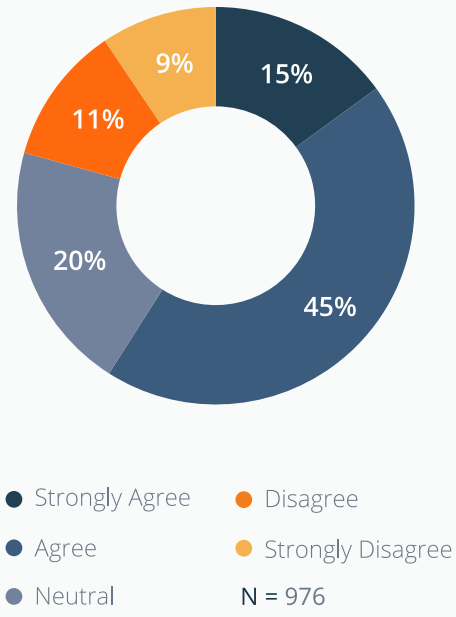
What are the business drivers that determined your team's need for a code review tool? *fig.9*



I often learn from others when I participate in code reviews. *fig.10*



My team uses code review to onboard and train new developers. *fig.11*



4 in 5 software professionals say they often learn from others in code reviews.

For the first time, we can put a number to how effective code review is as a learning tool, with 81% of respondents agreeing (50%) or strongly agreeing (31%) *(fig.10)*. No other question in this report elicited such a high response for "Strongly Agree."

Next, we asked whether teams are taking this learning approach and applying it to onboard and train new members *(fig.11)*.

3 in 5 development teams use code review to onboard new members.

While 81% of respondents said that they learn during code reviews, only 60% agreed (45%) or strongly agreed (15%) that their team uses reviews as a tactic for onboarding new developers.

The 22% difference between the two figures shows an increase of ~8% to neutral, ~8% to disagree, and ~7% to strongly disagree. Later in our Recommendations section, we will look at what impact using code review for onboarding has on a team's overall code review satisfaction.

Setting Code Review Guidelines

Last year, we added a couple of questions to get a better sense of how teams are setting expectations and tracking their success.

Expectations

Setting expectations can take many forms. It can be as informal as coming to a consensus on the definition of a term or as formal as using a custom checklist with time stamps.

In 2018, 73% of respondents said they know what they're supposed to be looking for when they review a teammate's code.

We updated this question around whether teams have set explicit guidelines on performing reviews. These are the 2019 results:

My team has guidelines for how reviews should be performed.

- **54%** of respondents agreed (41%) or strongly agreed (13%) that their team has guidelines on how reviews should be conducted.
- **25%** of respondents said that they either disagreed (15%) or strongly disagreed (10%)

While it isn't a perfect comparison, it looks as though the change in wording resulted in almost a 20% difference. While only 54% of respondents have guidelines for how reviews should be conducted, 73% say they know what is expected of them during a review. In our Recommendations section, we'll show analysis on whether it makes a difference to have guidelines or not.

Reporting and Metrics

Last year, we added a question to the survey to find out how many managers and teams are tracking their code review process with reports.

My team regularly pulls reports and metrics on our code review process.

- Only **29%** of respondents said they agreed (22%) or strongly agreed (7%).
- **49%** of respondents disagreed (24%) or strongly disagreed (25%).

More teams are not tracking metrics than teams that are. Year over year, the percent of teams pulling reports remained flat. In 2018, only 40% of respondents said that their team wasn't pulling reports on their code review process. The 9% change in this year's report represents a shift from "neutral" to "disagree" or "strongly disagree."

Despite continuous improvement being central to several working methodologies, including Six Sigma and Agile, most teams are not applying that approach to their code review process.

This could be a result of the decentralized review structure used by teams that only conduct ad hoc reviews or through tools like GitHub, Bitbucket, or GitLab. Those teams, if they're not using a dedicated tool, likely wouldn't have the metrics they would need to effectively report on their process.

In our Recommendations section, we'll show the impact that reporting on your process can have on a team's overall code review satisfaction.

The tools that a development team utilizes have a significant impact on the behaviors of that team. In this next section, we'll map the current tools and systems landscape across software development.

Section 3:
The Development Stack in 2019

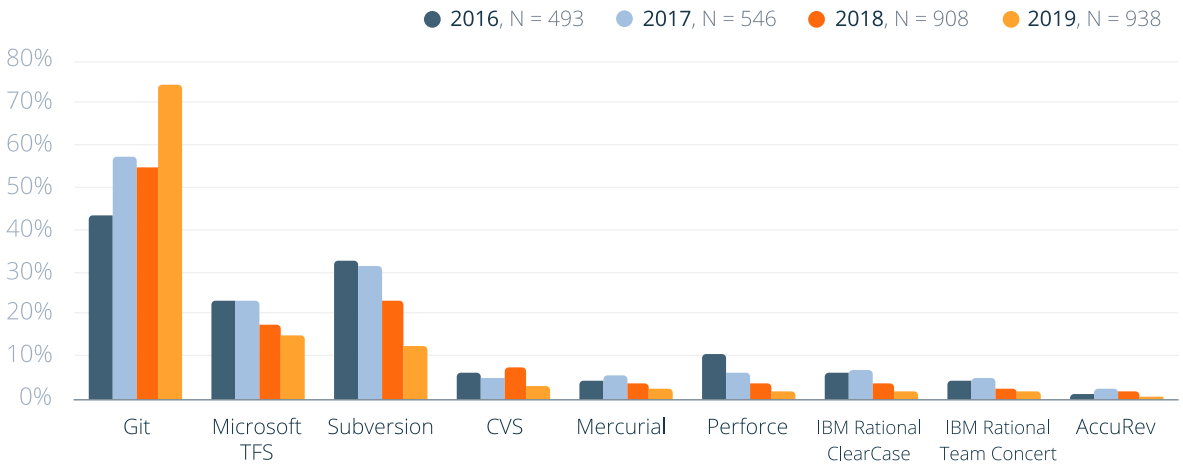
Within the same company, and sometimes the same team, developers utilize a diverse set of tools to do everything from project management to bug tracking to requirements management. In this section, we asked developers about the tools and systems their teams use.

Highlights:

- Git is increasingly the dominant source control management system, used by 74% of respondents. This is a **19% increase** compared to 2018.
- Bug tracking tools are the most common of any category, with **95%** of respondents listing a tool that their team uses.
- 64% of respondents are using GitHub, Bitbucket, or GitLab as part of their code review process. This is a **22% increase** compared to 2018.

Software Configuration Management Systems (SCMs) *(fig. 12)*.

Which software configuration management system (SCM) do you or your company currently use? *fig. 12*



3 out of 4 teams are now using Git for their software configuration management.

Git continues to be the most commonly-used SCM system for teams, used by 74% of this year's respondents. That's a steep jump from 55% in last year's survey. Microsoft's Team Foundation Server received the second most responses at 15%, down from 17% in 2018, 23% in 2017, and 23% in 2016. Subversion dropped to the third most popular system with 13%, a steep decline from 23% in 2018, 31% in 2017, and 33% in 2016.

10% of respondents selected that they were using no SCM system at all, similar to 11% last year. While the percent using SCM systems remained flat year over year, the shift to Git is clear since no other system grew in popularity and most declined significantly.

We also asked developers about what repository management tool they are using.

- **67%** of respondents are using GitHub (54%) or GitHub Enterprise (13%)
- **36%** of respondents are using Bitbucket (24%) or Bitbucket Server (12%)
- **31%** of respondents are using GitLab (23%) or GitLab Enterprise (8%)
- **20%** of respondents are using Microsoft TFS /Azure Devops
- **9%** of respondents are using no repository management tool

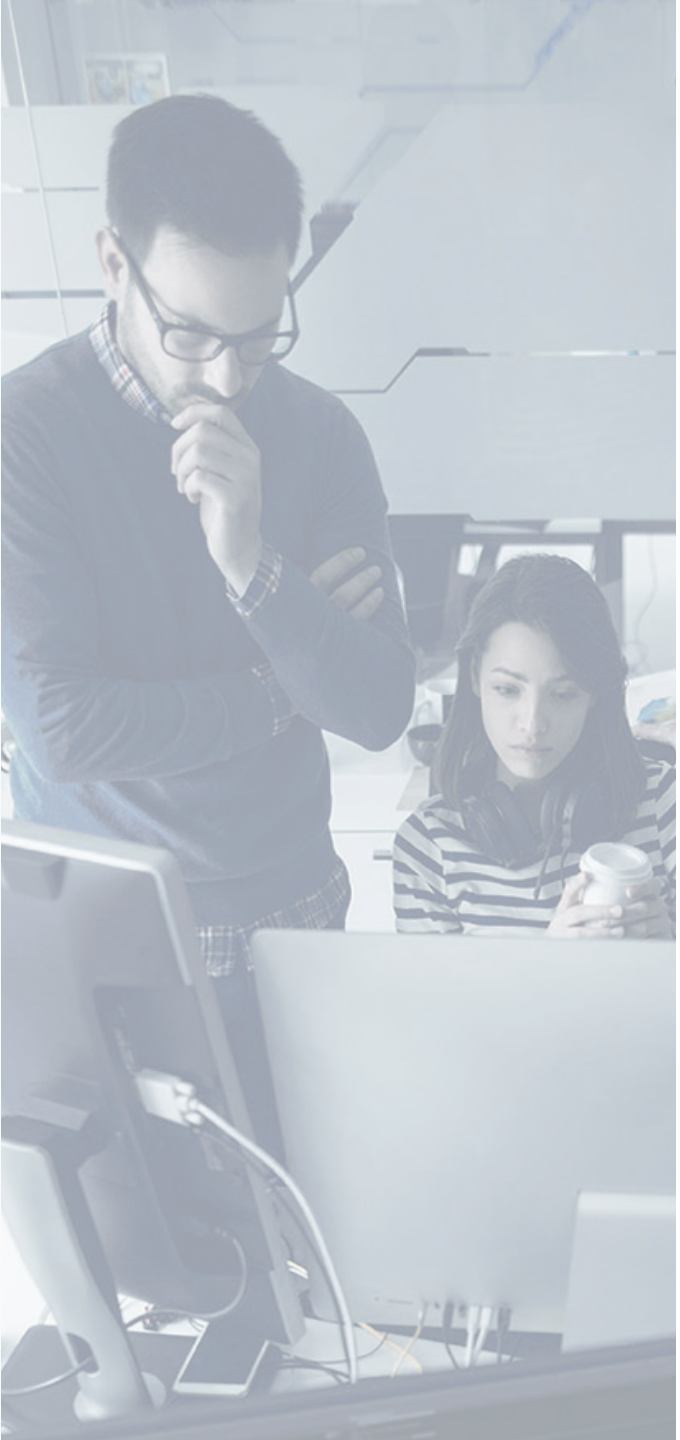
On average, developers using a repository management tool selected 1.5 of the options mentioned above, up from 1.4 last year. There has been a seismic shift within the last few years around the adoption of these types of tools. In our 2016 survey, 60% of respondents said that they were not using any of these tools. In 2017 and 2018, that number continued to drop to 34% and 30% respectively. Now in 2019, only 9% of teams are not using a repository management tool.

In last year's report, we noted that GitLab appeared to be the fastest growing tool in this area. In 2016, its offerings were used by 4% of respondents. In 2017, that figure climbed to 12%. In 2018, it had reached 18% market share. Our 2019 shows this dramatic trend is continuing with 32% of respondents now using GitLab or GitLab Enterprise.

GitHub made similar gains to GitLab from 2018 to 2019, gaining 13 points. The percent of respondents who are using Bitbucket or Bitbucket Server grew by 8% year over year. While the growth rates are different, the big picture story is that all of these tools gained in popularity, with some teams using multiple tools.

Integrated Development Environments (IDEs)

When we asked developers about what IDE they are using, the top 3 results changed for the first time in recent years, with IntelliJ taking the 2nd spot from Eclipse.



1. Visual Studio is the most popular IDE, used by **47%** of respondents.
2. IntelliJ rose substantially to the second most popular, used by **37%** of respondents.
3. Eclipse dropped dramatically to third, with **24%** of respondents.
4. XCode is used by **10%** of respondents, up 1% from the year before.
5. Netbeans is used by **7%**, down from 11% in 2017.

None of these tools had previously changed market share significantly in our 2017 and 2018 reports.

General IDE usage is basically flat year over year. Only 8% of this year's respondents said that they were not using an IDE, compared to 7% in 2018 and 8% in 2017. Our data also shows that the IDE market has a number of tools with much smaller user bases. 19% of respondents said that they were using an IDE not listed above.

Bug Tracking

There are many tools in the bug tracking market. Despite listing 13 options to select from, 14% of respondents selected "Other," mirroring the 12% of respondents who selected "Other" in 2018. From that trend, we can surmise that teams are using a wide variety of tools to achieve their bug tracking goals.

- Atlassian's Jira remains the dominant tool used by **65%** of respondents, up 10 points from 55% last year.
- Microsoft tools like Team Foundation Server (***14%***) didn't gain any market share for bug tracking year over year.
- Trello is used by **13%** of respondents, up from 8% in 2018.
- **The next tier of tools includes:** Excel (7%, down from 13% in 2018), BugZilla (5%), Redmine (3.8%), HP ALM (3.5%), Mantis (2%), Rational Team Concert (2%), Rally (1.4%), FogBugz (1.7%), Rational ClearQuest (1.1%), and YouTrack (1.1%).

Only 5% of respondents said that they are not using a bug tracking tool at all. That makes bug tracking tools more commonplace than any other tool category.

Requirements Management

Teams are currently managing requirements with many of the same tools used for bug tracking. Each respondent on average selected 1.9 tools from the options we provided, up 1.5 from 2018.

Most teams are using Atlassian tools for requirements management *(fig. 13)*. Jira is the top tool, used by 55% of respondents, followed by Confluence at 37%. The next four most prominent tools are all from Microsoft: Word (16%), Excel (14%), Team Foundation Server (13%), and SharePoint (6%).

HP Application Lifecycle Management (ALM) is used by 3.5% of respondents for bug-tracking purposes and 3.6% for requirements management. In 2018, those figures were 8.3% and 6% respectively.

A similar decrease applied to IBM's Rational tools. 3.1% of respondents said that they were using an IBM tool for bug tracking tools and 3.6% were using IBM's Rational DOORS or Rational DOORS Next Generation for requirements management. In 2018, those figures were both 5%.

Last year, Atlassian led both the "bug tracking" and "requirements management" categories with one tool, but Microsoft ultimately had a much higher

What tool are you using for requirements management?

fig. 13

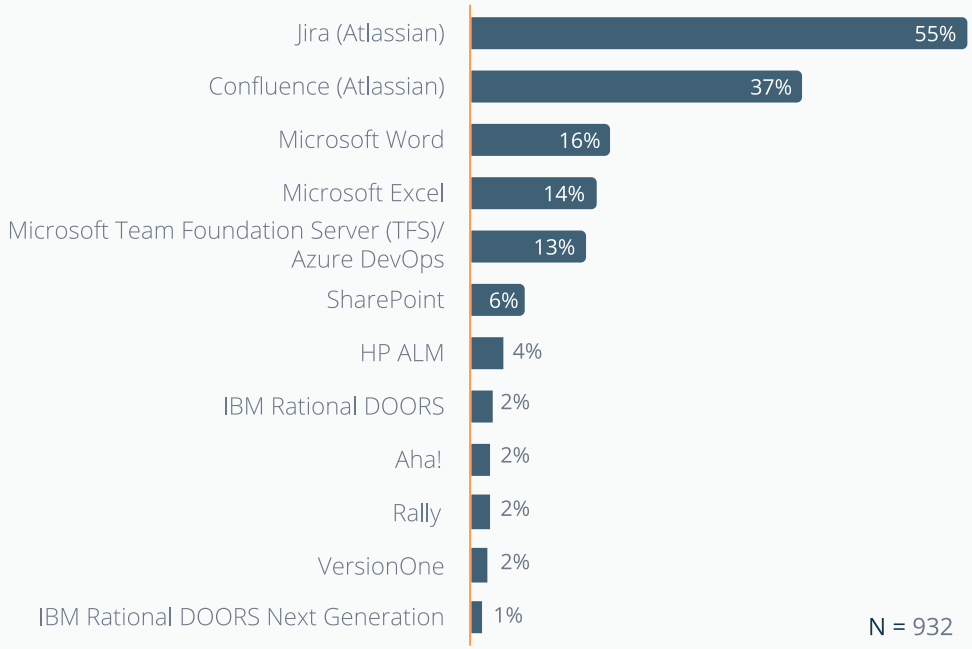
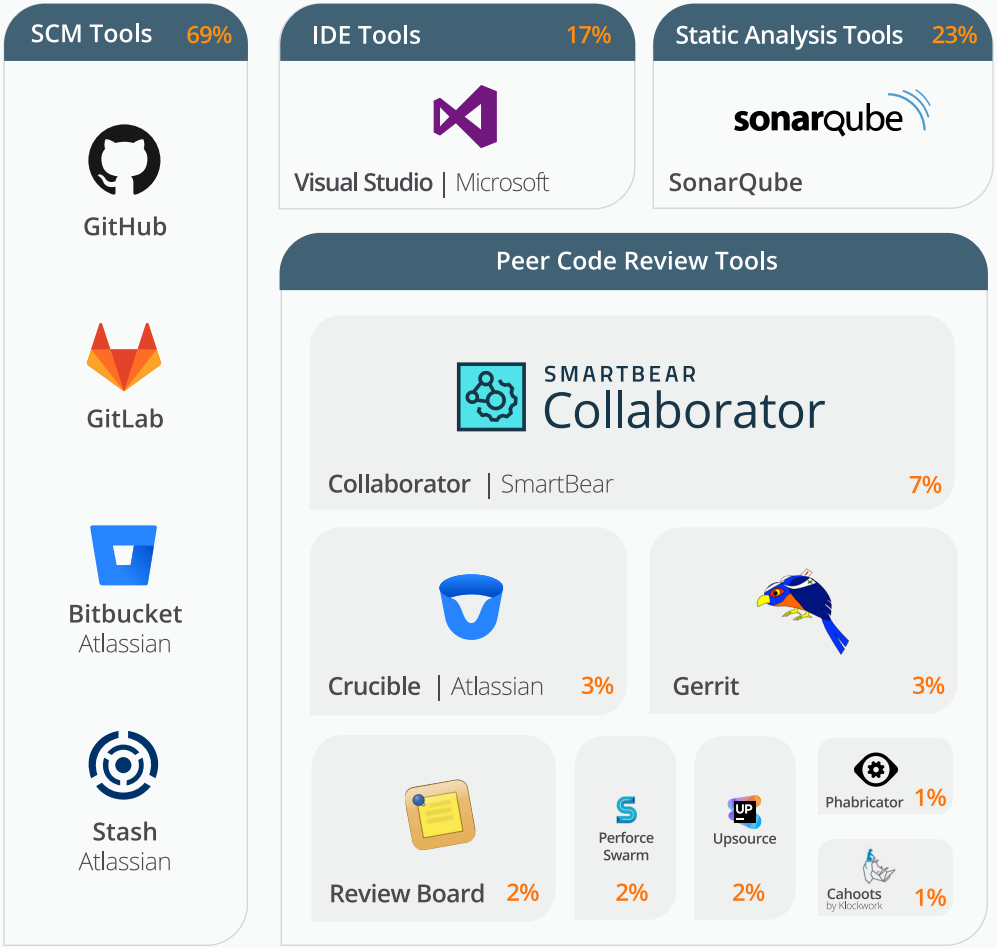


fig.14

N = 905



share of the market. Microsoft tools comprised 48% of all the total responses for this question, while Jira made up only 23% of responses.

This year, Atlassian eclipsed Microsoft in total market share for requirements management with 55% of the tools selected compared to 29% for Microsoft. Some of this change can be attributed to the addition of Confluence as an option in this year's survey. Even without the inclusion of Confluence, Atlassian's Jira would still be used more than all the Microsoft tools combined for requirements management.

Code Review

Given the primary topic of this report, understanding what tools developers use to conduct code reviews is critical. There are four segments of tools for this category: repository management tools, IDEs, static analysis tools, and peer code review tools.

- Repository tools like GitHub, GitLab, and Bitbucket enable developers to conduct reviews by sending pull requests to team members.
- IDEs like Visual Studio and Eclipse have varying degrees of review capabilities.

- Static analysis tools like SonarQube and Veracode enable general spot-testing of code.
- Peer code review tools like Collaborator by SmartBear and Crucible by Atlassian empower developers to review each other's code, identify bugs, and provide feedback.

In the fig. 14 visual, you can see that 69% of respondents use repository tools as part of their code review process, whether they're conducting reviews via pull request or integrating with a dedicated peer review or static analysis tool. This is up dramatically from 42% in 2018.

- Of the dedicated peer code review tools, Collaborator is the most prominent in the market, used by 7% of respondents.
- SonarQube is used by 23% of respondents. It is the only static analysis tool listed and was added to last year's survey because it continued to be mentioned in the "Other" category in years prior. No other static analysis tools have been popularly suggested.

In the Recommendations section of this report, we'll revisit the impact that using a code review tool has on overall code quality satisfaction.

Section 4:
Release Cadence and Team Firmographics

In this section, we'll look at release cadences, related challenges, the size and disposition of development teams, and how teams are represented across industries.

This is our first year including release cadences in this report. We added it as a data point to inform on changes in team size and ability to release on time, which are perennial questions.

Highlights:

- Development teams of 6-10 people are more common year over year, representing 41% of teams in 2019.
- "Changing Requirements" is the number one thing that prevents teams from getting their releases out on time.
- Monthly is the most common release cadence for development teams, selected by 26% of respondents.

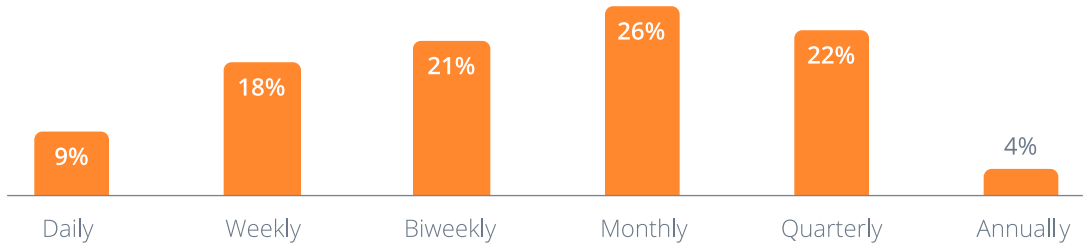
Release Cadence

For the first time, we asked our respondents to share information on how frequently their team puts out new releases *(fig.15)*.

The distribution shows a wide variety of team practices. Waterfall development is generally associated with releases that are on a quarterly cadence or longer. Agile development teams are aspiring to release frequently, often working in two-week sprints. While we didn't specifically inquire about what methodologies teams identify with, this data shows a bell curve.

How often does your team put out a new release? *fig.15*

N = 860



Even though teams may be "waterfall" due to contractual delivery or the nature of their work, many waterfall teams are adopting agile practices. The teams working with this hybrid "Wagile" approach may be more likely to release on a monthly basis, the most common cadence with 26% of responses.

Daily releases are likely a result of the recent adoption of continuous integration / continuous deployment pipelines (CI/CD). Many teams are using tools that support testing and build automation. Once configured, those teams are able to release on a daily basis. This approach is adopted most heavily by teams working on web or mobile applications.

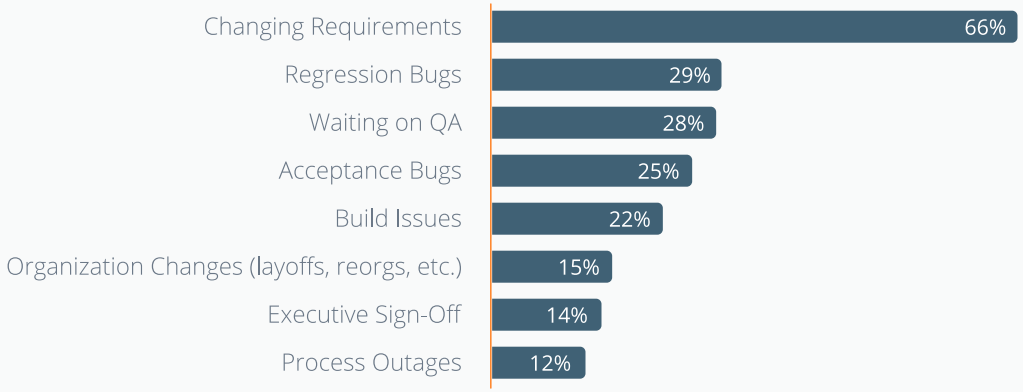
Release Challenges

This is our first year asking about what prevents teams from getting their releases out on time *(fig.16)*. In our Introduction, we highlighted the trend towards polarization on whether teams are able to get out releases on time. These are some of the challenges contributing to that trend.

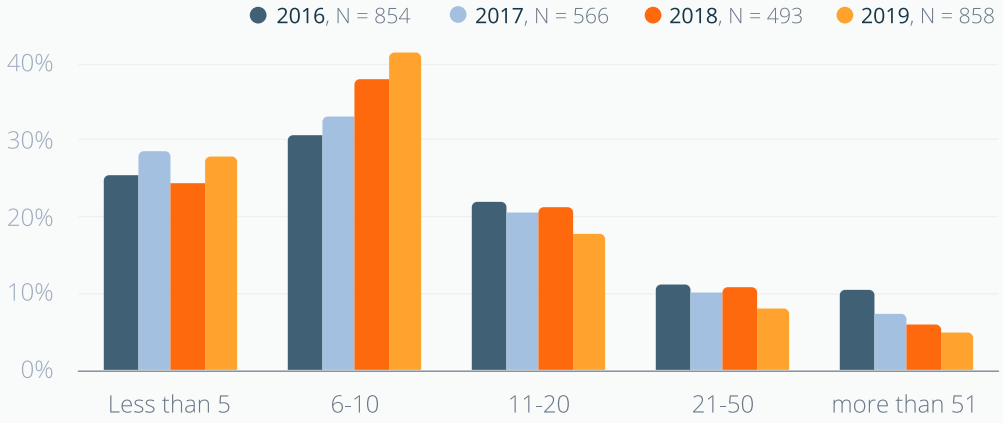
"Changing Requirements" is the #1 obstacle that prevents teams from getting releases out on time, selected by 66% of respondents. The next three reasons are "Regression Bugs" (29%), "Waiting on QA" (28%), and "Acceptance Bugs" (25%).

What most prevents your team from getting a release out on time? Select all that apply. *fig.16*

N = 858



What is the size of the development team you are on? *fig.17*



What is the total number of employees at your company?

fig.18

Number of Employees	Response Percentage
Less than 25	21.6%
26 - 100	22.6%
101 - 500	19.2%
501 - 1000	7.6%
1001 - 10,000	15.6%
10,000+	13.4%

Communication and collaboration are at the root of these top challenges. "Changing Requirements" might be the result of insufficient planning and design, or could be caused by miscommunication between business stakeholders and product owners. "Waiting on QA" could be the result of an insufficient handoff between developers and QA, or could be a sign of bandwidth challenges that might need to be solved with hiring or test automation.

Size of Development Teams and Companies

In order to communicate more effectively, many organizations are evaluating their team sizes. In the chart on the page prior, you can see how the size of development teams has changed since 2016 *(fig.17)*.

Teams with less than 5 people represent 28% of respondents. The most notable shift is the continued move towards teams sized between 6 to 10 individuals, comprising 41% of respondents in 2018. This is up from 38% in 2018, 33% in 2017, and 31% in 2016.

At the same time, teams of more than 51 people have been steadily declining, from 11% in 2016 to only 5% this year. Teams of that size are likely working on enterprise-grade projects, but the sheer volume would make it difficult to adopt modern development practices like scrums and retrospectives. Even if these large teams are developing in a waterfall framework, the attractiveness of smaller teams and agile tactics may be a catalyst for this shift.

We asked respondents about company size to know what data biases might exist in these trends *(fig.18)*.

You can see that the distribution across different company sizes is relatively even. So the trends and insights in this report should be applicable regardless of your own company's size.

Respondent Firmographics

Roles Surveyed

The first question of our survey this year was, "What best describes your role?" Overall, developers

comprised 48% of the respondents, followed by software architects (15%), development managers (9%), and testers (7%).

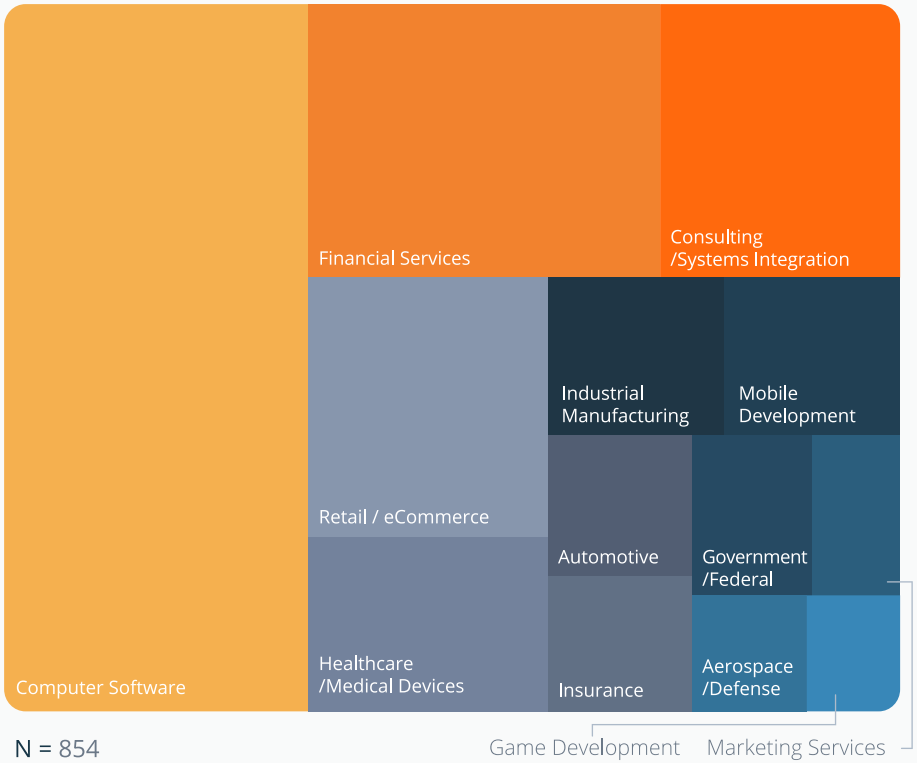
Industries Surveyed

In this year's survey, we heard from individuals working in at least 35 different industries. This is flat year over year. This chart shows the respondent breakdown by industry *(fig. 19)*.

Because of our wide audience of respondents, the findings of this report are applicable across many roles and industries doing software development. In this next section, we'll provide insights and recommendations that your team can act on to improve your code review process.

What industry do you work in? *fig. 19*

- Computer Software
- Financial Services
- Consulting / Systems Integration
- Retail / eCommerce
- Healthcare / Medical Devices
- Mobile Development
- Industrial Manufacturing
- Automotive
- Insurance
- Government/Federal
- Marketing Services
- Aerospace / Defense
- Game Development



N = 854

Section 5: Recommendations for Your Team

Every development team would like to be satisfied with their code review process. We isolated the key differences between teams who are dissatisfied or very dissatisfied with their code review process and those who are satisfied or very satisfied. For the sake of clarity, we removed respondents who were neutral about code-review-process satisfaction.

These are 5 recommendations that any team can use to improve its code review process:

1. Conduct code reviews on a daily basis.

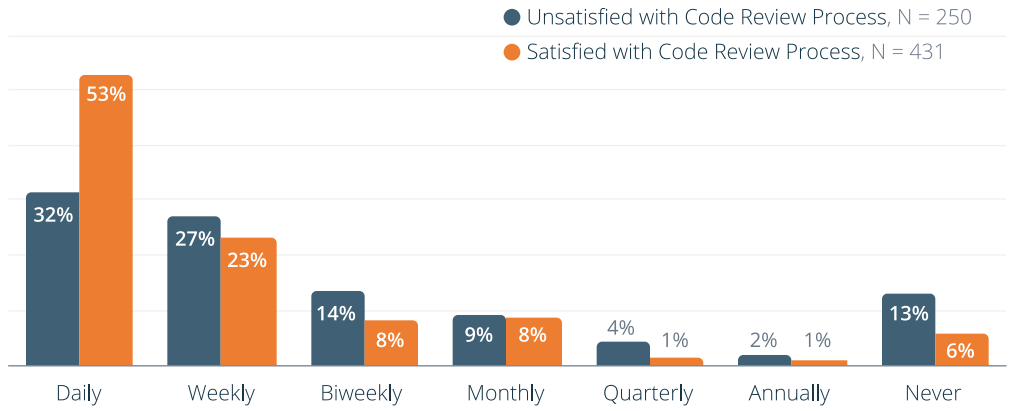
In this chart below, you can see the difference in review frequency between respondents who are satisfied with their code review process and those who are not *(fig.20)*.

Satisfied teams are twice as likely to perform code reviews on a daily basis (53%) compared to teams that are dissatisfied (32%). Last year, the disparity was similar at 45% and 22%, respectively.

While daily code reviews might sound disruptive to some developers, high review frequency translates to better code quality. There is a compounding effect when you introduce all the benefits of code review into daily behavior. Communication improves, knowledge about the codebase is shared, and fewer bugs make it through development to QA.

How often do you participate in any kind of code review process?

fig.20

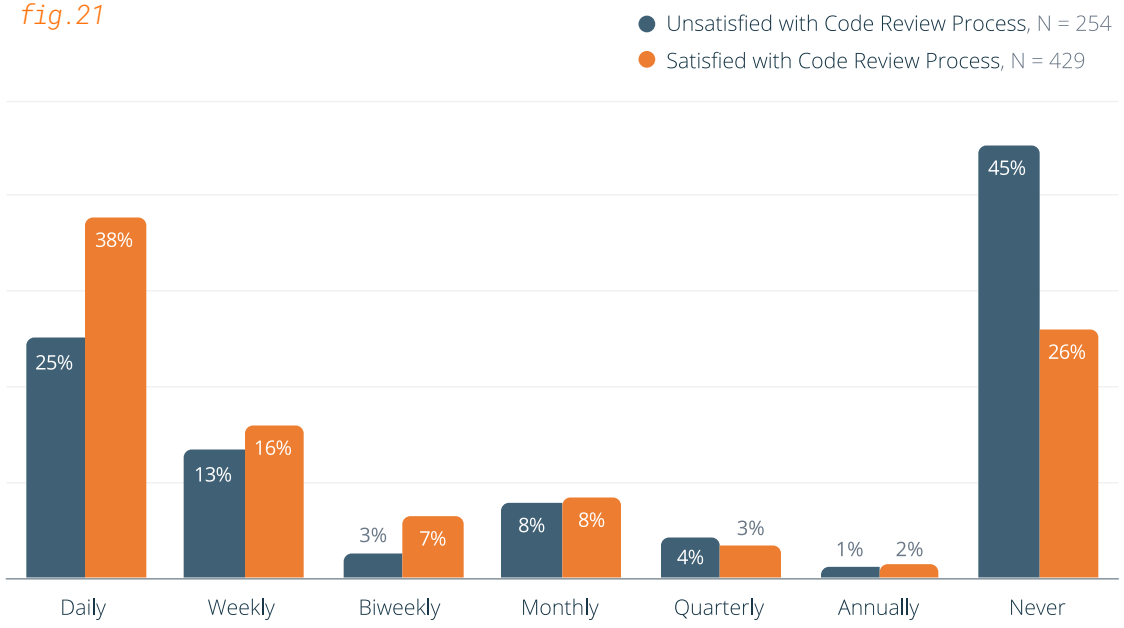


2. Conduct tool-based reviews.

The typical code review approaches are tool-based, ad hoc or "over the shoulder," and meeting-based. This year's report reaffirms that taking a tool-based approach makes a major difference on code review satisfaction *(fig.21)*. Teams that perform tool-based reviews are more likely to be satisfied with their overall code quality. Cohorted by code review satisfaction, 74% of satisfied respondents are conducting some kind of tool-based reviews. Comparatively, only 55% of unsatisfied respondents are conducting tool-based reviews. Based on these results, we highly recommend conducting tool-based code reviews. Your team is more likely to be satisfied with your code review process, and subsequently more satisfied with your overall code. If you're a developer on a team that needs additional budget or managerial approval to make this possible, share this report with them or read our **recent blog post** on how to quantify the return on investment in your code review process.

How often do you participate in a tool-based code review process?

fig.21



3. Set clear guidelines for your code reviews.

Earlier in this report, we mentioned that 54% of respondents say their teams have guidelines for how reviews should be performed. This chart below shows just how impactful guidelines can be in relation to code review satisfaction *(fig.22)*.

Teams with guidelines are twice as likely to be satisfied with their code reviews.

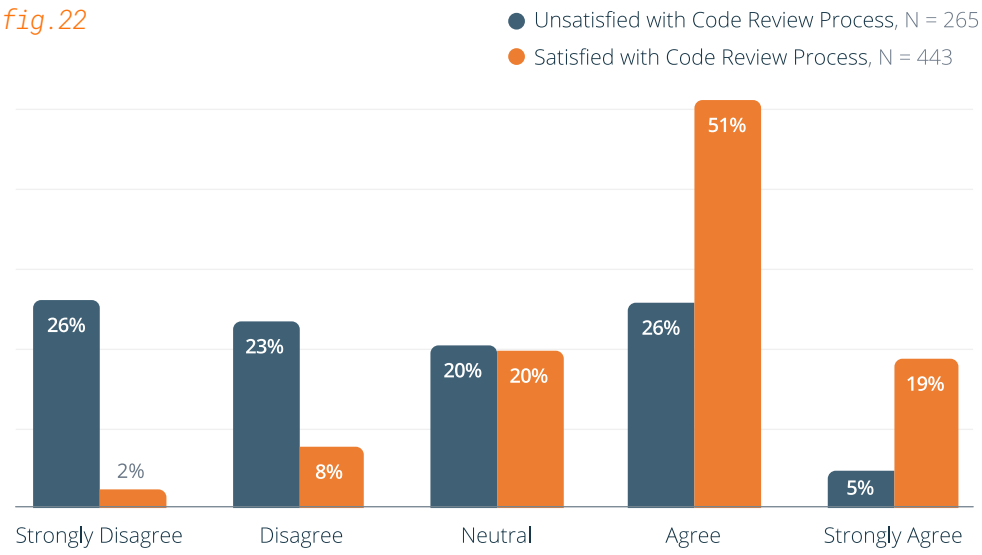
Cohorted by code review satisfaction, 70% of satisfied respondents have guidelines on how reviews should be performed. Comparatively, only 30% of unsatisfied respondents have guidelines.

The clearer you can set expectations, the more likely it is that your team will produce higher quality software. This applies to both code and document reviews. We mentioned earlier that the "satisfied code review" cohort performs reviews on more types of documents, and more often leverages a tool for their process than the "unsatisfied code review" cohort.

How can you ensure that your team is clear on expectations? Outline them in a checklist for both code and document reviews. Code review tools like Collaborator allow you to build custom checklists in review templates, so participants can easily see what's expected of them on each project.

My team has guidelines for how reviews should be performed.

fig.22



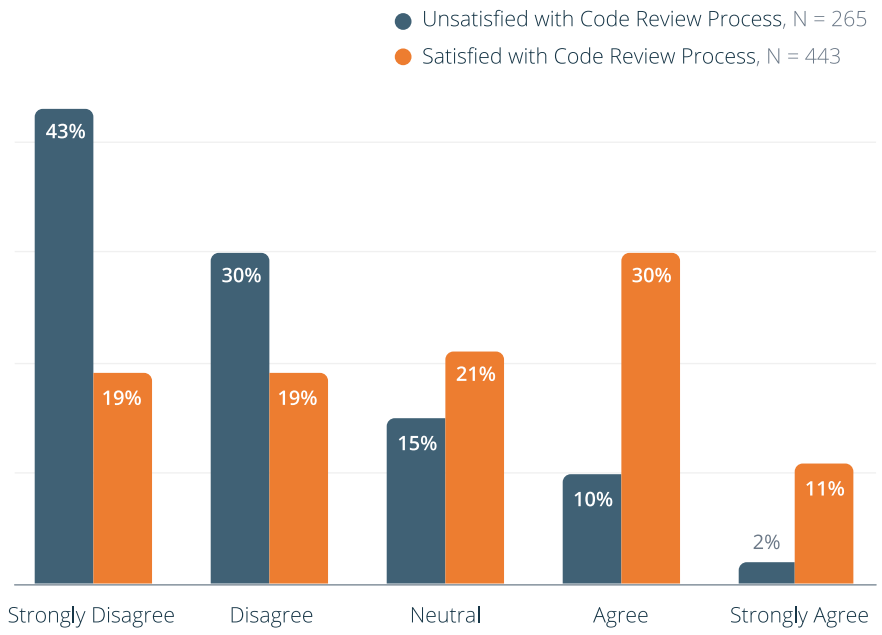
4. Choose review metrics to track and make reporting part of your process.

Teams that report on their process are 4X as likely to be satisfied with their code reviews (fig.23).

41% of satisfied reviewers (those who agree and strongly agree) regularly pull reporting on their process, nearly 4 times that of unsatisfied reviewers (12%). It is hard to be satisfied with your code review process when you can't track its aggregate effectiveness. Teams that do, through review reports and key performance indicators, are more likely to know what to improve when making changes.

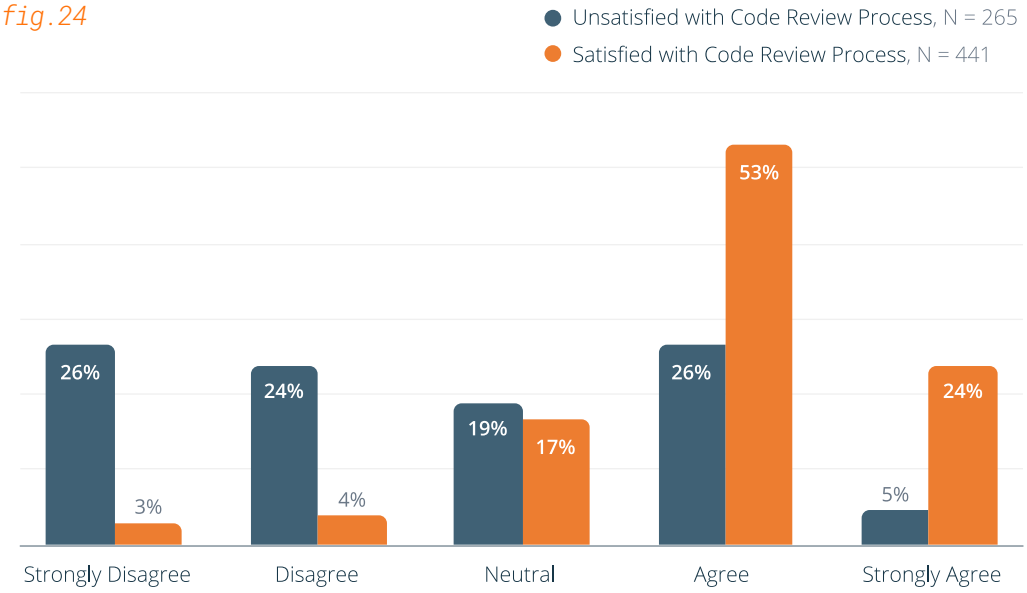
Adopting a tool that enables you to track key metrics and pull custom reports on peer code reviews is the fastest way to start driving meaningful process improvement.

My team regularly pulls reports and metrics on our code review process. *fig.23*



My team uses code review to onboard and train new developers.

fig.24



5. Use your code review process to train and onboard new developers.

We mentioned earlier how 60% of teams use code review as a tactic to train and onboard new developers. We found that those teams are taking the right approach.

This chart shows the breakdown between the two code review satisfaction cohorts *(fig.21)*.

Teams that use code review for onboarding and training are twice as likely to be satisfied with their code reviews.

77% of teams that are satisfied with their code review process are using it as an onboarding and training tactic, compared to only 31% of the unsatisfied cohort. 80% of all respondents say that they often learn during code reviews, so the pairing is a natural fit.

From onboarding as a junior team member to mentoring as a senior engineer, your code review process can serve as a daily learning and knowledge sharing vehicle. When your team is learning and collaborating on a daily basis, they'll build trust and improve your code quality at the same time.

Fostering a Collaborative Culture

Besides checklists and meetings, ensuring that your team is always comfortable and empowered to communicate is critical. Building a successful review process is about more than just adopting a tool, conducting daily reviews, and tracking metrics. It is about fundamentally empowering team members to own their code.

Just being conscious of how your team is communicating, from language to medium, can allow you to anticipate problems before they manifest. A culture of constructive collaboration can be the powerful undercurrent to propel your team's development to the next level, as long as you're willing to make the investment in your process.

Section 6: Handing Over the Mic

We've added a new section to our report this year. At the end our survey we asked a simple open-ended question:

What makes for a great code review?

Our respondents were generous enough to take the time to share their experience with us, so you deserve to hear from them directly. Here are just a few of the oustanding responses.

Jan S. says:

What makes for a great code review? The answer is the "Holy Grail" we are looking for. Almost every bug discovered in the production environment in last two years or so was due to mistaken or omitted code review. The requirements expressed by a natural language are often ambiguous, do not specify required behavior of the software in margin conditions and so on. You may have many tools to support continuous integration, code quality and coding standards conformance, but you must review the whole development process from its beginning, i. e. from the correct understanding of requirements to its materialization in the code. Otherwise you get "nice and conforming" code, that does something different than what has been required.

Jonathan M. says:

First, fostering an atmosphere of learning and growth. There needs to be a common perception of code reviews being an opportunity to grow as a developer, and to have a safety net, rather than an opportunity for criticism and admonishment. Second, sharing the "why" in addition to identifying the "what" that is the problem or the "how" to fix it.

Laurens H. says:

It depends on the context in which you are performing the review. A great code review for maintainability in JAVA might focus on code smells, method names and possibilities for refactoring. A great code review for security in C++ might focus on which libraries are being called how input is being processed and how pointers are being used. A good code review can be automated and performed with static analysis. But a great code review is dependent on people with expert knowledge and experience.

Emil N. says:

Getting value from code review is all about fostering a culture of open questions delivered in a polite and professional format. If reviewers are reluctant to ask questions, it's a complete waste of time. On the other hand, if the author experiences the review process as a personal crucible, it'll lead to team friction. It's all about carefully establishing code review as a team cooperation effort where everyone is interested in honest feedback to improve their code and their coding practices. It takes a while to establish, and it takes effort to maintain, but it's essential to making code review worth anything. If you're not asking questions, you're not developing an understanding of the code and the problems it's solving.

Mark M. says:

Having a review done by people who know what to look for and have a reasonable expectation of understanding the code and its impact. A number of data points collected by a number tools are required for such a complete picture and code alone is not enough. Understanding how the code effects the process as a whole is an important part that would provide more of the context needed to fully assess.

Christopher O. says:

Setting aside the time to go through the changed code properly, rather than just skimming over it. Consider what was meant to be changed, and whether it works as expected. Asking questions of the code author, rather than demanding changes. Having an understanding between members about how much back-and-forth should be allowed, and being clear that minor or cosmetic issues don't necessarily have to be fixed.

Myreli B. says:

We started our code review practice to mostly improve code quality, assure design patterns were being properly followed and create a collaborative environment in a more natural way. Our results were almost instantly noted, as everyone accepted the habit and more experienced developers gained more space (without a hierarchy structure being imposed) to share best practices and code guidelines. Besides that, less experienced developers felt comfortable enough to share its own experiences and doubts, and the team grew more united. What I mean is, what makes for a great code review is great engagement and an important reason behind: code improvement and team engagement.

Simon M. says:

A good mix of senior and juniors reviewing code, leaving questions and observations, not just ticking the box. Ideally people from other teams not just the team you are on who have a less vested interest in just saying yes. We have also found a template explaining the PR and what it about are helpful to avoid unnecessary work by reviewer to understand what it is about. Also QA build generated on PR as sometimes it helps for a dev to just see what the built version looks like as well.

Bryan R. says:

A great code review answers two questions: (1) Does the code do what it is supposed to do? (2) Is the code easily understood and maintained? The first reminds reviewers they need to familiarize themselves with the feature or defect involved and in so doing they also become familiar with more of the overall software application. While the word "easily" in the second is quite subjective, it reminds reviewers to look at every level for unnecessary complexity and unanticipated side effects.

Robert P. says:

Reviewing small chunks of code with proper unit tests written greatly improve the code review process. Great code reviews will tell you why you want to make the changes being suggested, not just that you should make the changes. Any company coding standards should be well documented and frequently seen mistakes or broken standards should be addressed to the whole team instead of just kept between the developers working on the review.

Mike F. says:

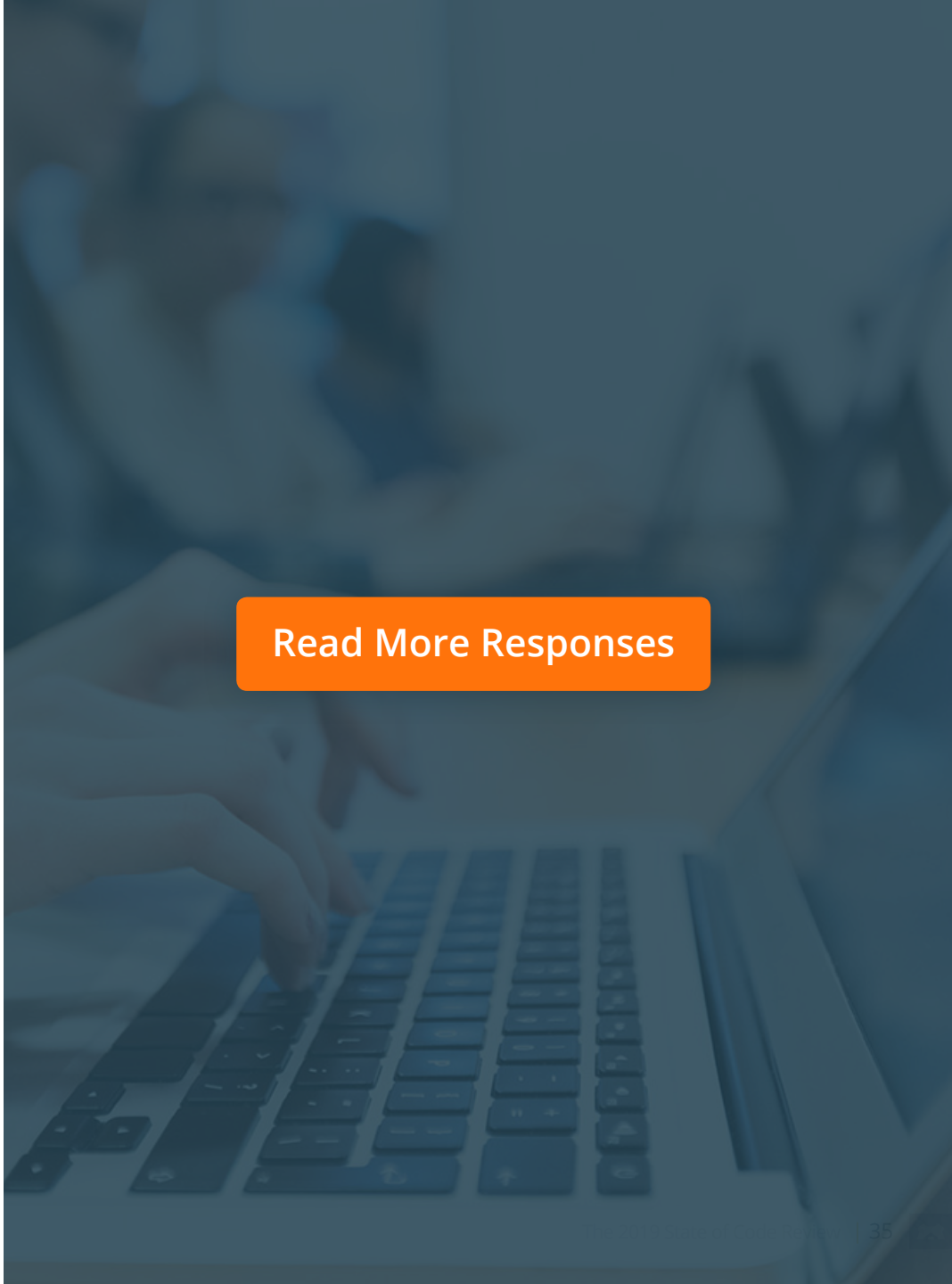
1. Grooming a story as a team in advance - knowledge sharing and understanding expectations of advancing the story to 'Done'.
2. Empowering all members of the Dev team (developers and QA) to write tasks, that identify specific buckets of work to complete the story, during Sprint Planning. This helps to share knowledge and provides additional insight into the "how" of the implementation.
3. Sharing that a story is ready or nearing 'Review' status during the daily scrum.

Vanessa S. says:

The best code reviews are those that are provided by a colleague who has obviously taken the time to write out thoughtful comments, and make helpful suggestions. They take time.

Robert P. says:

Reviewing small chunks of code with proper unit tests written greatly improve the code review process. Great code reviews will tell you why you want to make the changes being suggested, not just that you should make the changes. Any company coding standards should be well documented and frequently seen mistakes or broken standards should be addressed to the whole team instead of just kept between the developers working on the review.



Peer Code & Document Review

Build *Trust* in Your Code

Explore Collaborator


SMARTBEAR