The 2020 State of Code Review:

Trends and Insights into Collaborative Software Development

SMARTBEAR

The 2020 State of Code Review Report is presented by SmartBear

We provide tools to help you with API quality, collaboration, monitoring, and more.

Pro Tools

SwaggerHub

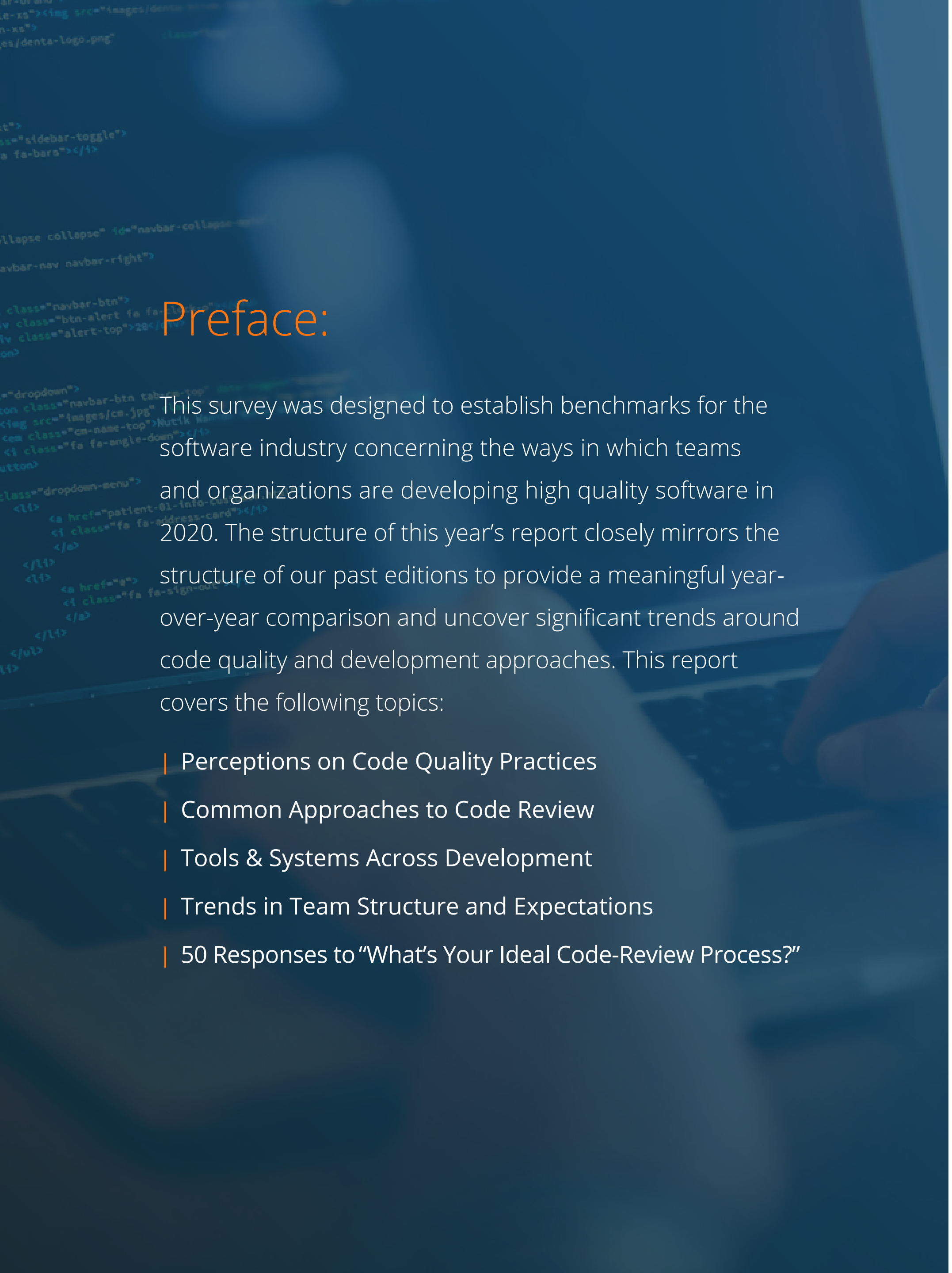
ReadyAPI

Collaborator

Swagger

Open Source Tools

SoapUI



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 2020. 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
- 50 Responses to "What's Your Ideal Code-Review Process?"

Methodology:

SmartBear Software conducted a global online survey over the course of ten weeks during the months of June and July 2020. The findings are based upon aggregated responses from more than 740 software developers, testers, IT/operations professionals, and business leaders across 20 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 fewer 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.

For the 2nd year straight, 2 out of 3 respondents are satisfied with their code quality.

With all of the information we've gathered this year, there's of course the underlying, "How has this been affected by the pandemic?" No one can say for sure – especially since our survey went underway in the middle of it – but the trends we see are that teams are collaborating more remotely because, well, they have to.

For some companies, COVID-19 has accelerated a remote-first culture. With it comes greater demand for various synchronous tools such as Slack, Zoom, and Google Meet.

We've seen a similar trend in demand for Collaborator. With a large majority of people working remote, meeting-based and ad-hoc (over the shoulder) reviews aren't an option. In addition to allowing teams to collaborate remotely, Collaborator is also a workflow tool, ensuring a code- and document-review process is being followed without having to watch it, per se.

Quality and satisfaction are up.

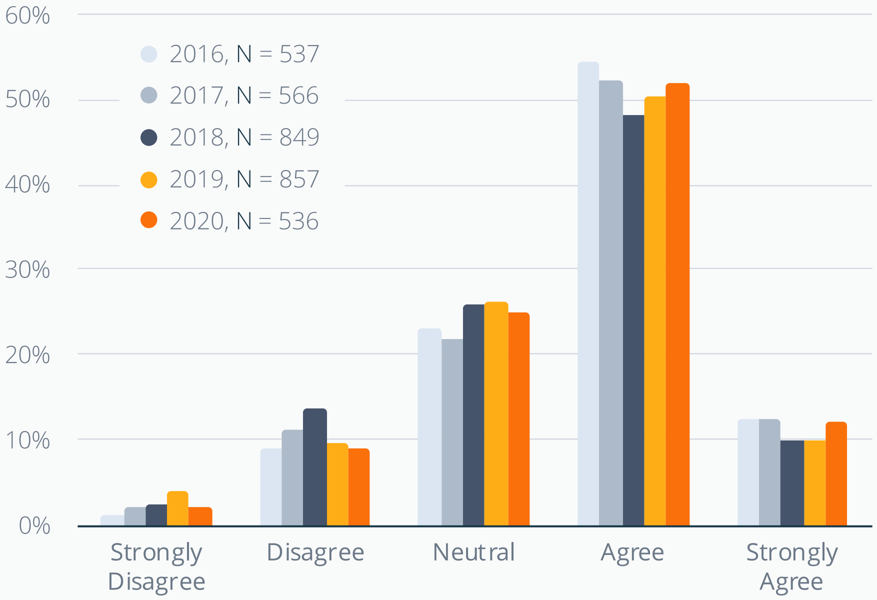
Last year, we noted an upward trend in satisfaction with software quality. This year, satisfaction has once again gone up: 64% of respondents reported being satisfied, making it a 6% increase over the last two years. There's also a noticeable drop in dissatisfaction.

Over the last 5 years, at least 55% of respondents indicated they're happy with the quality of software they help deliver. It's encouraging to see that more than half of people are satisfied with the solutions they're delivering. It speaks to the sense of ownership people have for their contribution and their desire to ship quality products.

This trend tells us that less teams are sacrificing quality to meet deadlines. It's safer now to assume that businesses know rushing out unfinished product is not a viable, long-term strategy. So their natural conclusion (along with ours) is to invest in better processes, because it helps ensure deliverables with higher quality.

I am satisfied with the overall quality of the software I help deliver (specifically regarding performance, bugs, etc.)

fig.1



Collaboration is helping quality go up.

For the second year in a row, 2 out of 3 respondents are happy with the quality of the software they deliver. This is based on 65% of survey respondents stating they're either satisfied or highly satisfied. Only 11% are unhappy with the quality of software compared to 14% last year.

Over the last 5 years, at least 55% of respondents have indicated they're happy with the software quality. In 2018, there was a drop in satisfaction, with only 58% of respondents being satisfied. But over the last two years, this has increased by 6%. Our assumption is that, with more and more teams moving to Agile, they're automating processes and collaborating more effectively *(fig.1)*.

Companies are making (and missing) deadlines at the same rate as before.

The results of our next question correspond with the prior question's percentages. 65%, or about 2 out of 3 respondents, told us their company is regularly able to get releases out on time. Only 12% of respondents indicated that their company often misses deadlines *(fig.2)*.

Since 2016, the number of respondents that say their company often misses release deadlines has increased by 1%, while those who regularly meet deadlines has gone down by 1%. So, there has not really been any change to the success rate of releases going out on time. For those that are neutral, it begs the question that half their releases are on time, and perhaps half are late.

Also, year over year, each grouping (***Strongly Agree*** through ***Strongly Disagree***) is rather consistent. It makes us wonder if those that aren't able to get releases out on time are doing anything about it. It remains to be seen if the problem lies in things like personnel or general communication.

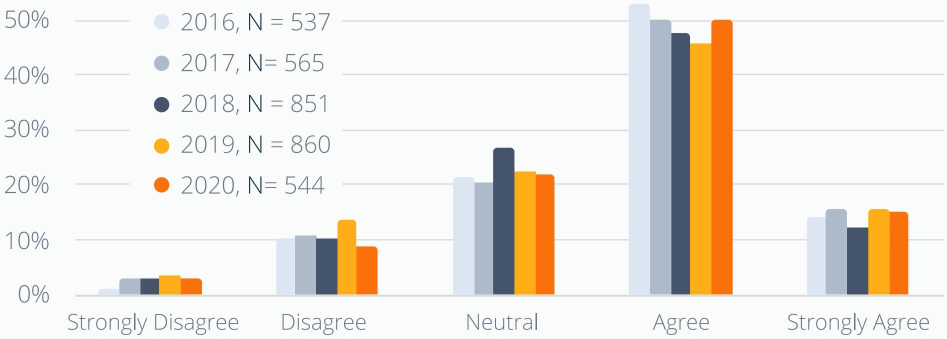
Half of respondents are satisfied with their code review process.

As for satisfaction with their team's current code-review process, 1 out of 2 respondents are at least satisfied *(fig.3)*. Almost a quarter (23%) of respondents are not. Since 2018, we've seen a 6% increase in code-review process satisfaction. This is great news – people are collaborating through their code-review process, and overall satisfaction with the quality is increasing as well.

Over the last 3 years, there's been a 4% decrease in the amount of people dissatisfied with their code-review process. This is a positive trend, even if they only feel neutral about the change. However, the numbers suggest they're changing camps entirely, and are landing in "satisfied" territory.

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

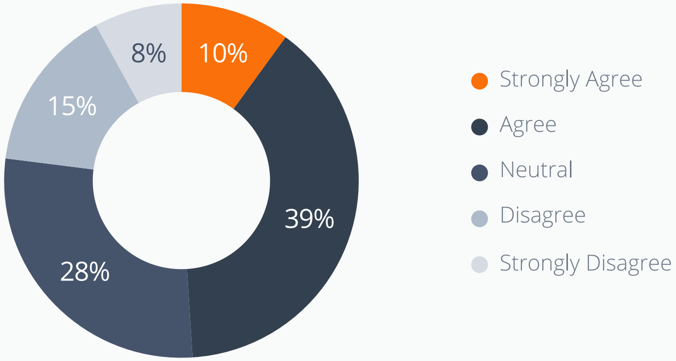
fig.2



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

fig.3

N = 669



Number 1 way to code quality? *Review.*

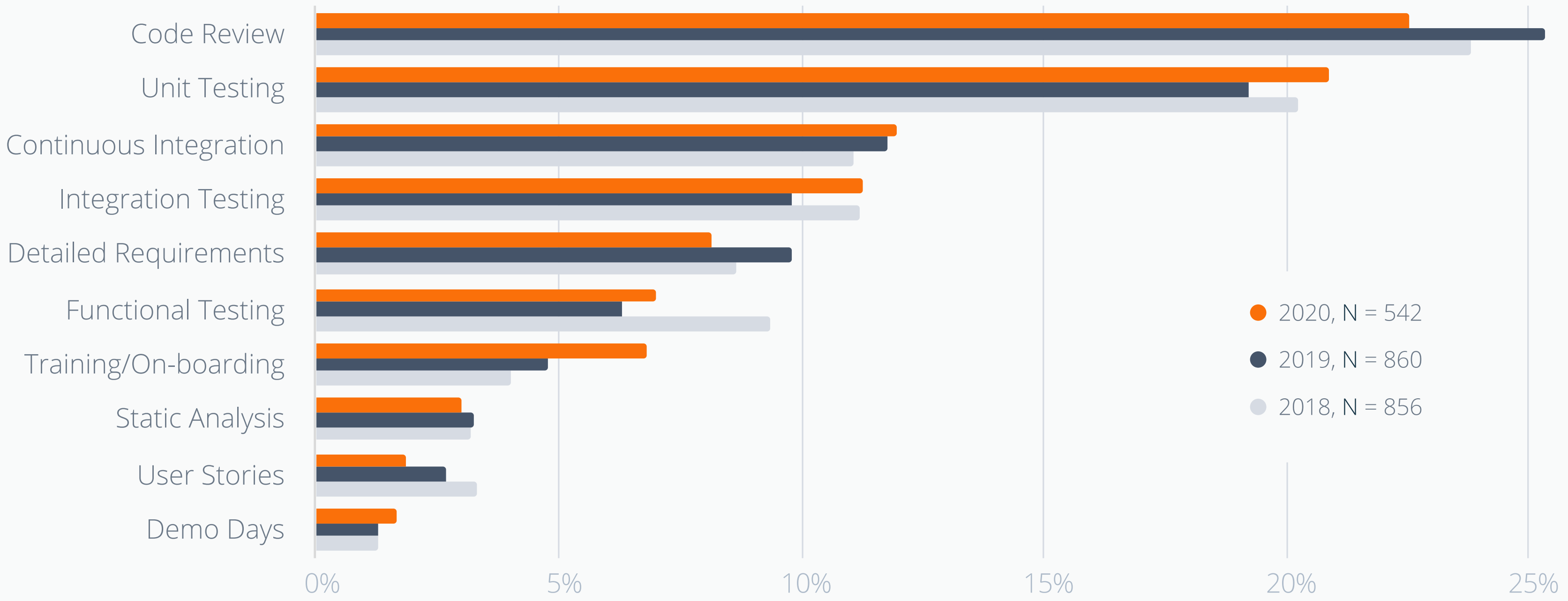
Over the last 3 years – and somewhat expected – our respondents have told us that the number one way a company can improve code quality is through ***Code Review***. This year, 24% of our respondents indicated this.

Results also indicated that ***Unit Testing*** is the second most important at 20% of responses, followed by ***Continuous Integration and Continuous Testing***. As can be seen by the chart, unit testing is closing the gap on code review as the number one way to improve quality. This could be due to the fact that unit tests can be automated, and therefore catch issues without the involvement of other people. Regardless of which way is number 1, it's important for teams to have a multi-faceted approach to quality.

In 2020, we saw a jump in the importance of ***Training/ Onboarding*** in order to improve code quality. Since 2018, this has increased by 3%. We saw a slight decrease in the value of code review, dropping 3% over the last year *(fig.4)*.

Also over the last 3 years, the importance of ***Functional Testing*** has dropped 3% for our respondents.

What is the number one thing a company can do to improve code quality? *fig.4*



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

		Software Quality Satisfaction				
	N = 533	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
Code Review Satisfaction	Strongly Agree	0%	4%	11%	36%	49%
	Agree	0%	4%	15%	69%	12%
	Neutral	0%	6%	41%	47%	6%
	Disagree	3%	18%	32%	46%	3%
	Strongly Disagree	26%	29%	21%	21%	2%

There's a clear correlation between code review satisfaction and code quality satisfaction.

This chart shows the correlation between how happy respondents are with code review, and the satisfaction in overall software quality *(fig. 5)*. It proves to be a handy way to see how code review can directly impact software quality.

Over 80% of respondents who were satisfied with their code review process were also satisfied with the overall quality of their software. This is a similar percentage compared to last year's report, indicating an unchanged opinion that code review is important in delivering quality software.

Satisfaction with one's code review process was the most likely indicator of overall software quality satisfaction. Further on in this report, we'll explore year-over-year changes in code-review practices to identify common traits that contribute to a better peer-review approach.

Section 1: The Code and Document Review Process

New code-review approaches coming to the forefront.

The topic of "code-review approaches" was a multi-facetted question asked several times throughout the survey, so we have several takeaways.

Roughly 63% of respondents participate in some form of code review, at least on a weekly basis.

When it came to frequency and which approach used, 27% of respondents cited tool-based code review on a daily basis, and 19% noted weekly basis *(fig. 6)*. This is up 4% from last year. For ad-hoc reviews, 15% participate on a daily basis, and 29% weekly. This is unchanged from last year. With the exception of tool-based reviews, reviews are performed more often on a weekly basis by our respondents.

More bi-weekly reviews than before.

Over the last 5 years, respondents indicated they participate in biweekly reviews much more than before.

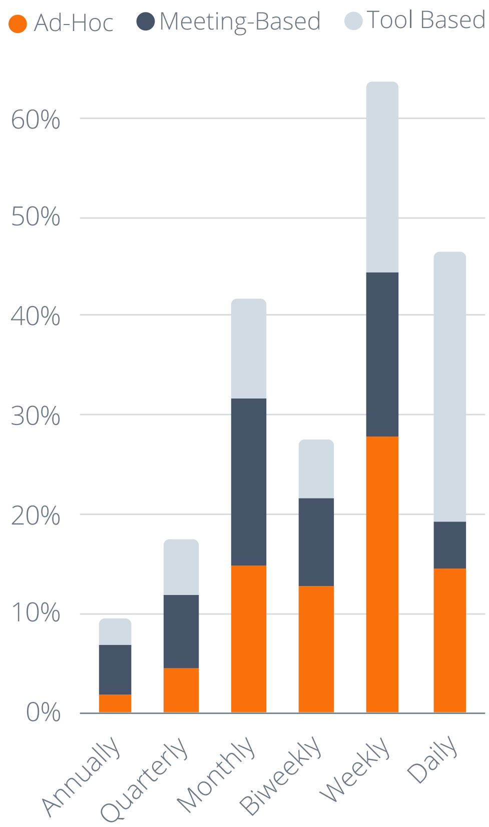
- Ad-hoc reviews 13% more
- Meeting-based 9% more
- Tool-based 6% more

Since last year, there was an average increase of 3% in tool-based, meeting-based, and ad-hoc monthly peer reviews.

In our 2016 survey, nearly 40% of respondents said they never participate in tools-based code reviews, while 47% said they never participate in meeting-based code reviews. Even for ad-hoc reviews,

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

N = 733



28% said they never took part. All of these approaches to code review have increased in popularity by roughly 9% over the last 5 years.

Weekly reviews most popular, but less of them are meeting based.

When all three approaches are looked at together, our respondents have shown us that a weekly peer-review cadence seems to be the most popular approach.

When looking at the data, it's not surprising that the number of people that never do meeting-based reviews is higher than the number of people that never do ad-hoc and tool-based reviews. Meeting-based reviews are often seen as more time consuming than the other forms.

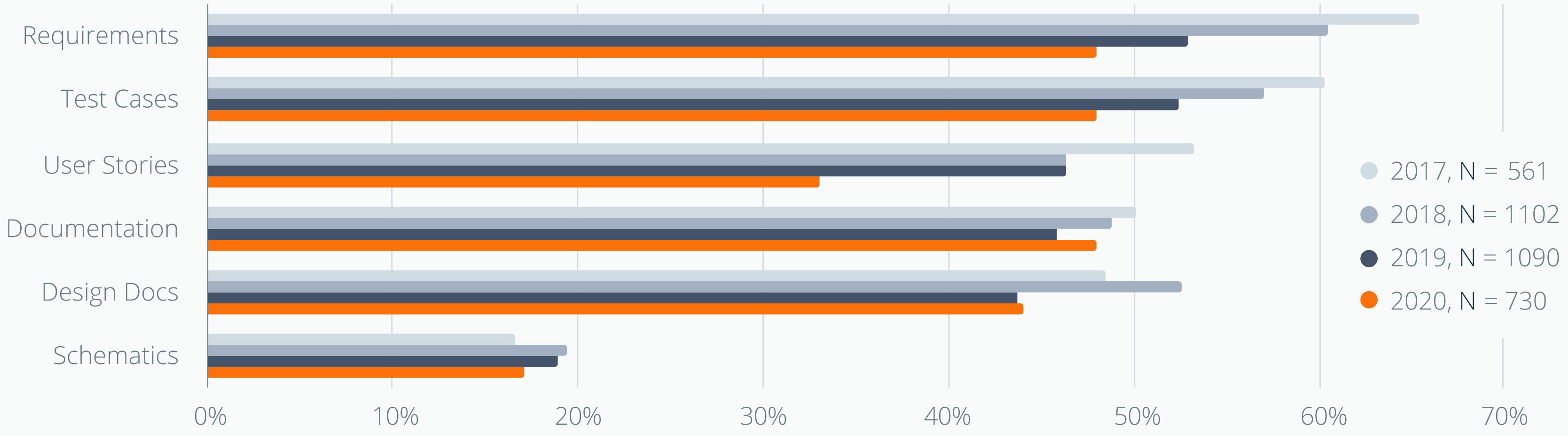
Additionally, many teams find it difficult to get everyone together in a meeting room due to distributions introduced by the global pandemic. In fact, many teams were forced to suddenly transition and learn how to collaborate remotely and across multiple time zones. Tools like Zoom and Microsoft Teams do make meeting-based reviews, though virtual, a feasible option, especially with the increase in remote workforces.

Artifact reviews are generally going down.

Over the years, the review of artifacts has generally trended downwards. This may be due to the transition away from waterfall workflows to more Agile software development practices that encourage iteration. People may have the perception that artifact review is not as critical, since they're allowed (as defined in the Agile manifesto) to respond to change instead of following a plan *(fig. 7)*.

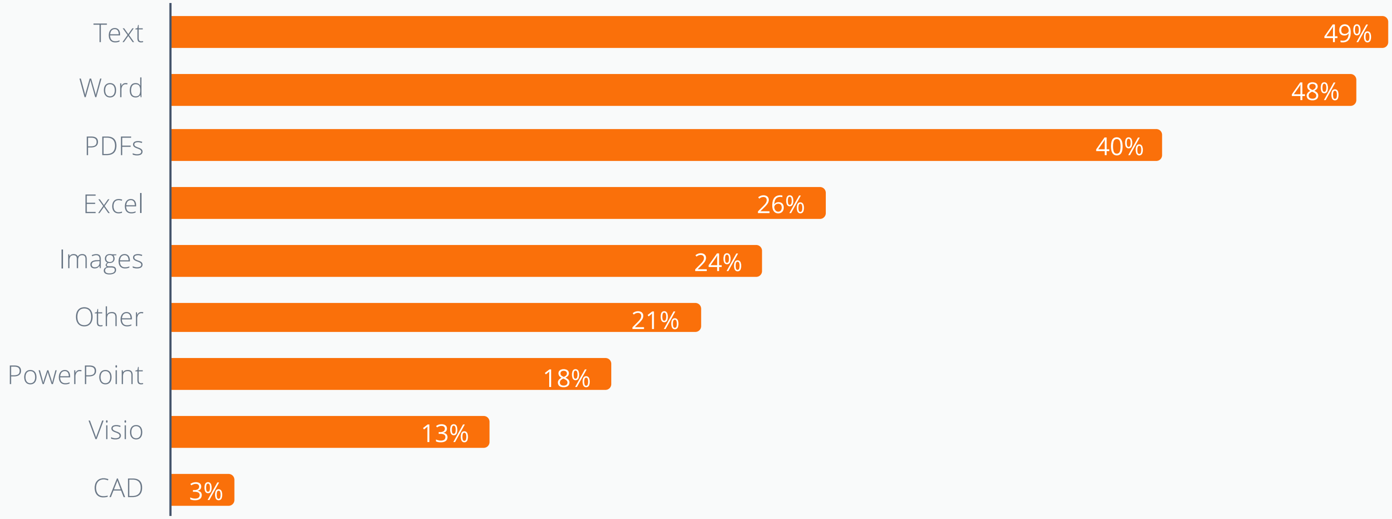
Out of all our respondents who do code review, 90% review artifacts. Exactly 1 out of 2 of those respondents

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



Which of the following document types do you review? *Select all that apply* *fig.8*

N = 669



are reviewing ***Test Cases***, ***Requirements***, and ***Documentation***. Over the last five years, respondent answers have indicated they've been reviewing ***User Stories*** 20% less – 13% of which dropped in the last year. It's hard to know why there has been such a significant drop, considering the growth of BDD.

Test Cases and ***Requirements*** are also down about 4% since 2019, and down over 12% since 2016.

We've seen a decline over the last few years across this entire list of artifacts being reviewed, though ***Design Docs*** and ***Documentation*** saw a small bump from 2019 results.

To dig a little deeper into the document review process, this year we added a new question about what specific document types respondents came across to review.

Out of the 90% of respondents that participate in at least some type of document review, almost half of them review Text and Microsoft ***Word*** documents *(fig. 8)*. The percentage of ***PDFs*** under review came in at 40%.

Respondents also indicated that they review ***Confluence Wiki Pages***, ***Google Docs***, ***Swagger Design Specs***, and more.

How do you conduct document reviews?

When the question of "how" came up, 1 out of 2 respondents revealed they conduct document reviews in meetings, and about 30% participate in them "over the shoulder," so to speak.

Currently, 43% of respondents conduct document reviews in a native tool like Confluence, Microsoft Word, or Microsoft PowerPoint, and 32% use a peer review tool for their document reviews. It'll be interesting to see how these percentages will change going forward.

Data suggests artifact review is declining, but is it?

In all, the year 2020 saw a few additional document-review questions added to the survey. Though the data suggests that artifact review (other than code) is generally declining, what's surprising is that that's not what we're seeing at SmartBear. We find that a lot of both new and current customers are coming to us to learn more about document review.

What isn't surprising is the types of documents people are reviewing, or that they're performing these document reviews in either meetings or in the native tool itself. With the latter, many are finding that the lack of process is making it challenging to ensure documented quality reviews are taking place.

Section 2: Perceptions on Code Review

An effective code-review process is key to ensuring the long-term quality of the code base – both from a defect perspective and for readability. As individuals, we experience code-review benefits differently than that of the business as a whole. This section explores this by looking at common benefits, use cases, and business drivers.

Improved Software Quality has been the #1 benefit of code reviews since we started this report five years ago *(fig. 9)*. This was followed by ***Sharing Knowledge Across the Team***, and the ***Ability to Mentor Less-Experienced Developers***.

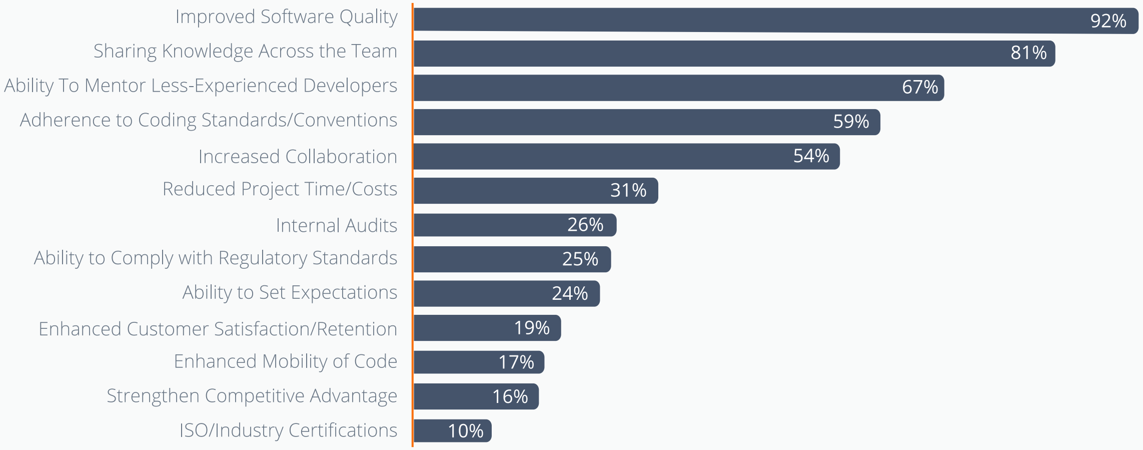
Since 2016, we've seen a 12% increase in the importance of mentoring less-experienced developers, and a 7% increase in knowledge sharing across the team. Teams continue to value knowledge sharing above external benefits. Such an investment by teams in long-term individual improvement is encouraging. It creates more productive and efficient teams, with better results down the line.

Also noted in the past five years, ***Adherence to Coding Standards/Conventions*** has risen 7%. ***Strengthen Competitive Advantage***, ***Enhanced Mobility of Code***, and ***Enhanced Customer Satisfaction/Retention*** have

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

Select all that apply *fig.9*

N = 735



generally declined as reasons since 2018 *(fig. 10)*. Though not as popular, they're still legitimate reasons for many teams, and presumably secondary benefits from the more popular benefits cited on this chart. Also note that ***Internal Audits*** have increased in importance by 6% since 2019.

What's great to see about code review is that regardless of the reason why you or your team might do it, this shows that **many teams are getting onboard with the idea** – and that there are numerous seen and unseen benefits they're getting out of it.

Because the use of code tools has grown so much, the inherent question of *why* you would you do it is what prompts us to ask this each year.

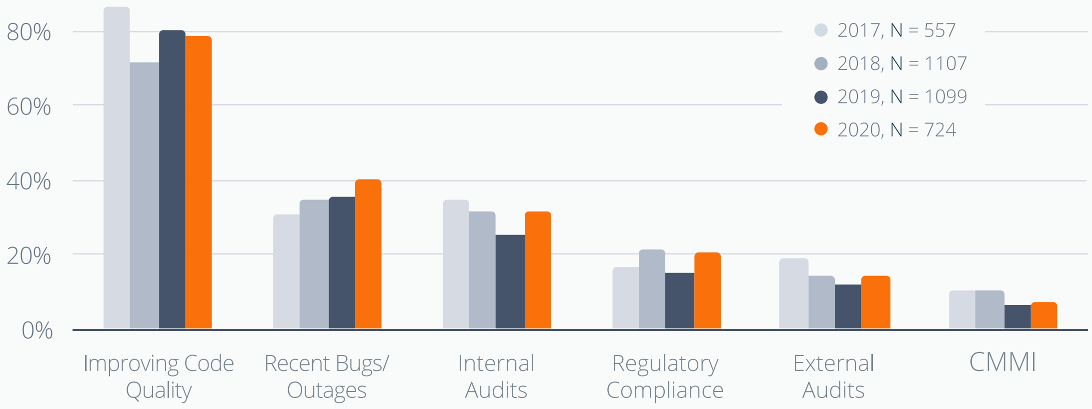
And with that, ***Improving Code Quality*** is the number one business driver when determining a need for a code-review tool, and has been number one for five years in a row *(fig. 11)*. Although we noticed a slight drop (1%) since last year, overall it's grown by 10% since 2016.

Most notably, ***Recent Bugs/Outages*** have grown significantly in importance since 2016, up 33% since then, and up 4% since last year. ***Internal Audits*** have also become more important, up 7% in the last year and 24% in the last five.

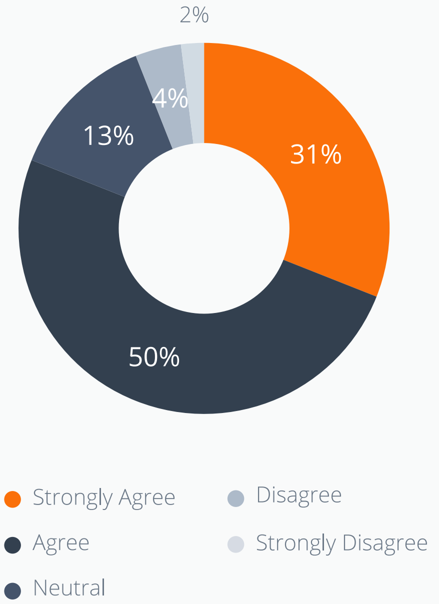
Benefits of Code Review (Gaining/Losing Attention) *fig.10* N = 735

Benefits of Code Review	**2020**	2019	2018	Net Change
Gaining Traction				
Ability to Mentor Less-Experienced Developers	**67%**	66%	57%	+10%
Sharing Knowledge Across the Team	**81%**	82%	73%	+8%
Increased Collaboration	**54%**	54%	52%	+2%
Improved Software Quality	**92%**	91%	90%	+2%
Losing Attention				
Enhanced Mobility of Code	**17%**	17%	26%	-9%
Enhanced Customer Satisfaction/Retention	**19%**	19%	26%	-8%
Ability to Comply with Regulatory Standards	**25%**	23%	31%	-6%
Reduced Project Time/Costs	**31%**	28%	37%	-6%

What are the business drivers that most determined your team's need for a code review tool? *Select all that apply* *fig.11*



I often learn from others when I participate in code reviews. *fig.12* N = 674



Why the increase across the board.

We continue to see record-setting levels of software-caused recalls across industries like automotive, healthcare, and consumer devices. Throughout 2020, software defects have impacted items as varied as airports, cars, parking meters, video games, and satellites. Additionally, new cybersecurity challenges are forcing teams to address vulnerabilities with more caution and nuance than ever before.

With the push to release software more frequently, for many teams it's come at a high cost: quality. Though code review is seen as a critical component to improving quality, several other initiatives are seen that way, too, including unit testing, integration testing, etc. It's important for organizations to focus on an area of weakness, improve on it, and then tackle the next. As the saying goes, "If you try to chase two rabbits at once, you won't catch either." The same could be said for attaining software quality.

4 in 5 respondents agree that they often learn from others when they participate in code reviews.

This has been one of the questions with the highest ***Strongly Agree*** rate since 2019. In fact, when considering this question, only 6% of reviewers indicated that they *don't* frequently learn from others during code reviews *(fig. 12)*.

Strengthening teams and individuals.

One of the benefits to peer review is our ability to share knowledge. However, it's important to remember that sharing knowledge isn't about telling people how to do something, but instead about coaching. Encouraging team members to do their own research, or providing them with a reference for how something could be done differently, is a great way to pass along knowledge and coach without you being the "expert."

This helps reviews be a little more impartial and can keep tension down between team members. After all:

> "*If you're not learning from reviews, what other source is providing you that unique perspective?*

In other avenues of code review's benefits, we once again asked respondents if they use it to help bring new employees up to speed. 60% of teams that participated agreed that, yes, they use code review to train and onboard new developers *(fig. 13)*.

Looking at it further, only 20% of teams do not leverage code reviews in this way.

This year, we also wanted to see how training and onboarding new developers with code review correlates with satisfaction of software quality.

We've found that software professionals who are using code review for onboarding developers are 50% more likely to be satisfied with their software quality. Respondents who use code review to onboard devs are 75% satisfied with the quality of software being delivered.

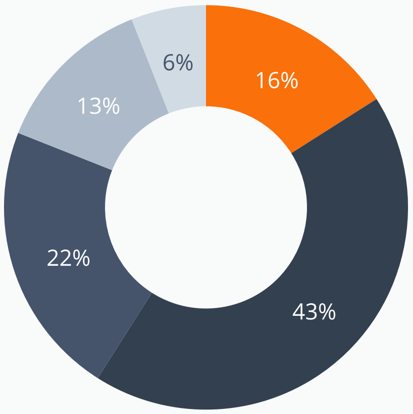
Alternatively, those who are satisfied with code quality fall to 50% if they do not use code review for onboarding and training new developers.

Digging further into each teams' operations, respondent answers show that only 35% of teams regularly pull reports and metrics on their code-review process. Even though it shows more teams are not tracking than are, this year it's up 7% since 2019, so trending upward. 40% of teams don't pull reports or metrics regularly, which is down 8% since last year.

On a higher level, it appears that 55% of teams have guidelines for how their reviews should be performed. This is up versus 2019 reports, but only slightly. Less than 25% report to having no guidelines.

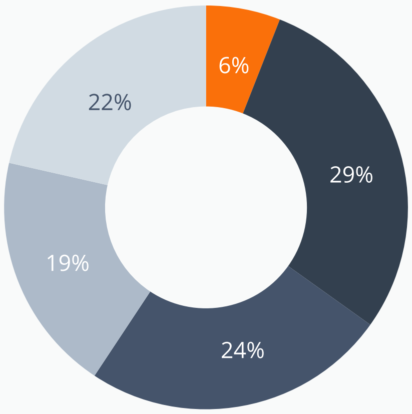
My team uses code-review to onboard and train new developers. *fig.13*

N = 673



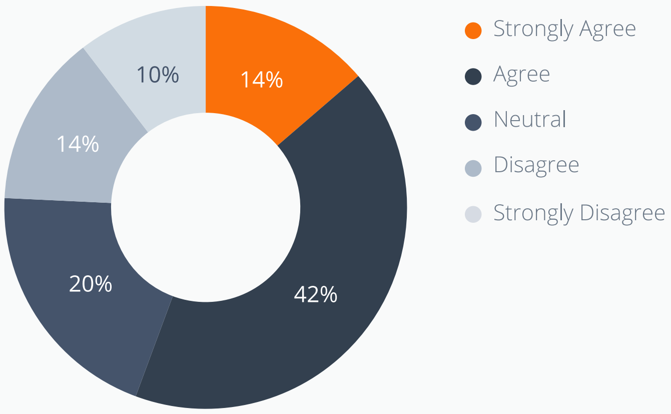
My team regularly pulls reports and metrics on our code-review process. *fig.14*

N = 671



My team has guidelines for how reviews should be performed. *fig.15*

N = 672



Process is the key – despite how you feel about it.

Guidelines for how reviews should be conducted are critical to teams and individuals getting the most out of their reviews *(fig. 14)*. Though most of us are not big fans of process, it's still a key element to driving consistency and quality. Your code-review guidelines don't need to be heavy handed; they can be a short checklist to remind you of some fundamental items to look for in a review. Things even as simple as including both a junior and senior developer, as well as QA team member, on every review.

Pulling metrics is a great way to see where you might have areas for process improvement. Reports can inform the team of trouble areas, or show the types of issues that keep cropping up during reviews *(fig. 15)*.

Section 3: The Development Stack in 2020

Git is the #1 used version-control tool.

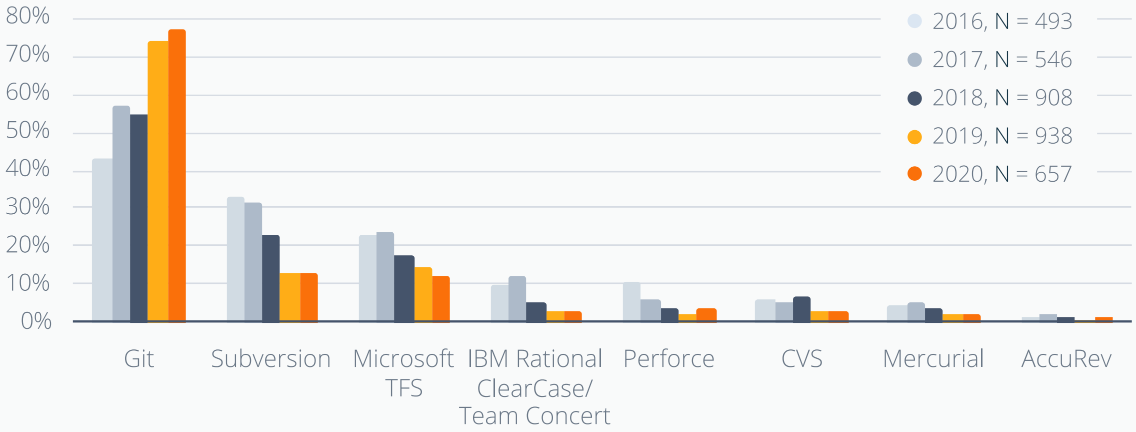
The answer to the below question was overwhelmingly ***Git***, which pulled away from the pack in 2016 and has increased its lead every year. This year it reaches an all-time high at 77%, followed by ***Subversion*** at 13%, and ***TFS*** at 12%, the latter two having taken a hit in recent years *(fig. 16)*.

Git has trended up over 33% in the last 5 years, with a 3% increase in the last year. Also in that span of time, ***Microsoft TFS*** has seen an 11% decline as Git has grown. ***Perforce*** also with a 6% decline since 2016.

Nearly 10% of respondents indicated they use no SCM system. This could be caused by some respondents focusing on document reviews, or simply be individuals or small teams that don't believe they need one. Another possibility is that some respondents are not actively using one because of their job function. To put it in perspective, the audience of the survey breaks down as follows: at least 50% developers, 12% software architects, and 8% development managers.

With the popularity of GitHub, GitLab, Bitbucket, Azure DevOps and others growing, it's not surprising that Git continues to see growth year over year while others decline as they're replaced.

Which software configuration management system (SCM) do you or your company currently use? *Select all that apply* *fig. 16*



Jira (Atlassian) is the #1 used tool for requirements management since 2016.

When it comes to requirements management, 1 out of 2 respondents made it known they use *Jira*. Jira has grown by 20% since 2016, but curiously saw a slight downward trend in our survey this year *(fig. 17)*.

Confluence is a close second with 36% of software developers using it, while *Microsoft* products – including Word and Excel – have grown 5% since last year. *Microsoft Team Foundation Server (TFS)/ Azure DevOps* has seen a 2% decrease since 2019. We'll see if that continues to trend down.

Other responses included *Trello*, *Google Docs*, *GitHub*, and *Asana*.

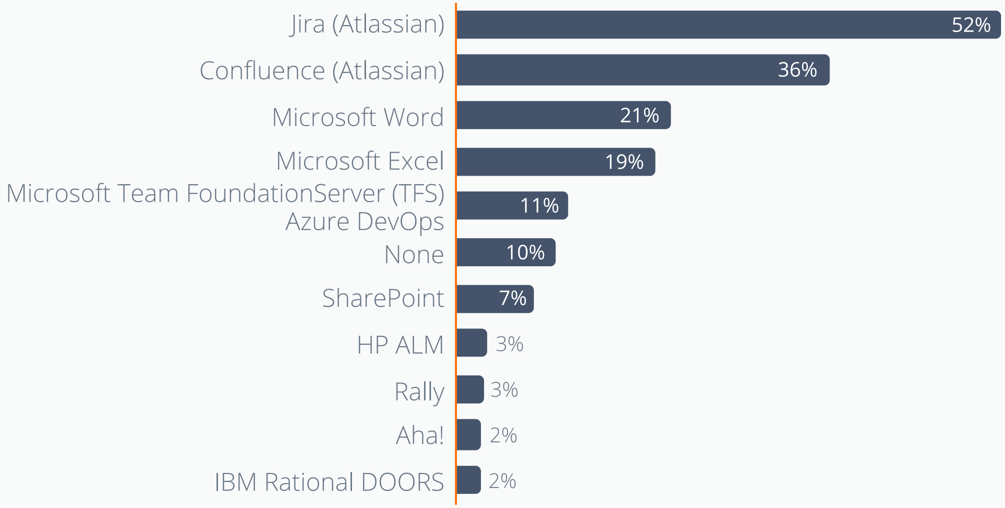
The adoption of repository hosting tools continues to grow.

It shouldn't come as any surprise that the use of repository hosting tools continues to grow. As more and more companies move away from

What tool are you using for requirements management?

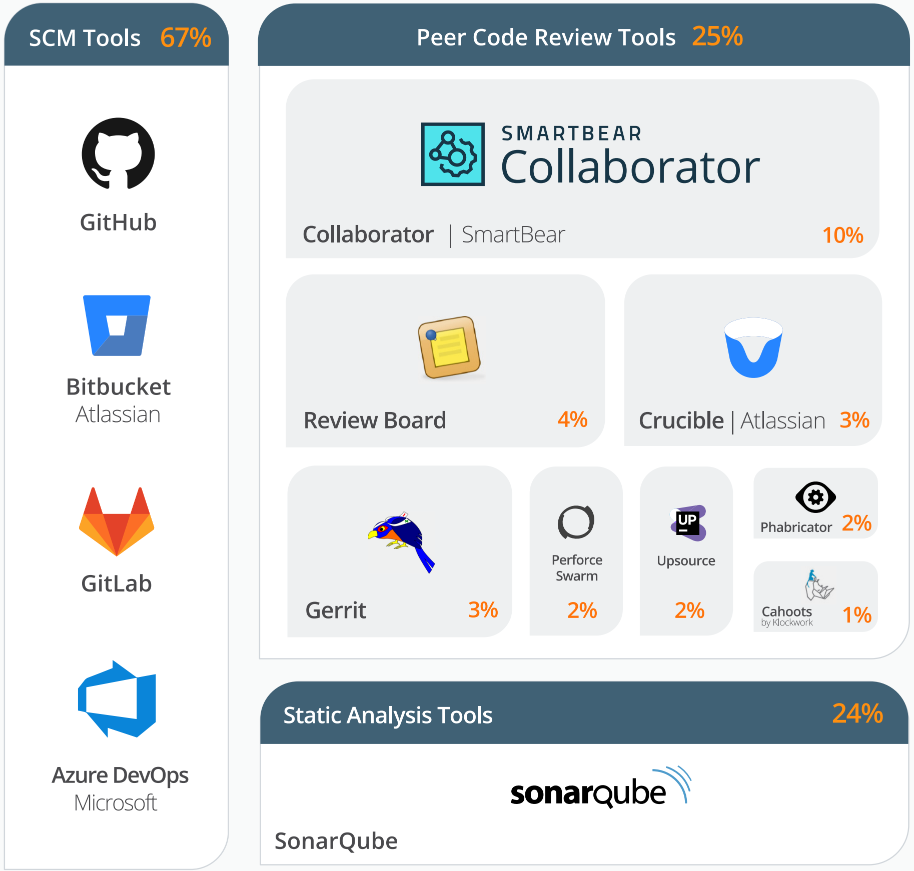
Select all that apply *fig.17*

N = 652



Within your department, do you currently use any of the following tools for code review? *Select all that apply* *fig.18*

N = 660



legacy-version control tools such as CVS, SVN, and others, Git is the natural choice. With that, many organizations see value in moving to a repository hosting tool as part of that transition.

The benefit of transitioning to a repository hosting tool is that it comes with a number of features, one of which is the ability to perform some level of code review. With this built-in component, some teams and organizations can do all the code review they need through these tools. In this year's survey, 67% of respondents said they use (***GitHub***, ***GitLab***, or ***BitBucket***) for code review *(fig. 18)*.

For peer review tools, which 25% selected in the multiple-choice question, 10% reported using ***Collaborator***. About 25% of people reported using static analysis tools for code review. Though static analysis isn't quite peer code review, it is a key review tool that teams should have in their toolbox.

Since 2016, the adoption of repository hosting tools including GitHub has grown by 43%, growing 4% since last year. Also since last year's report, the percent of respondents indicating they ***Don't use any tool for code review*** has gone down 3%.

Section 4: Release Cadence and Team Firmographics

Most new releases come monthly.

When asked how often new releases go out, 28% of teams reported that they deliver *Monthly*. Deliveries of *Quarterly*, *Bi-Weekly*, and *Weekly* were just behind, leaving a relatively wide gap before *Daily* and *Annually* (fig. 19).

Since 2019, we've seen growth in the number of teams releasing on a quarterly and monthly basis, while those delivering on a daily basis has decreased by 2%.

Requirement changes are clearly in the way of timely releases.

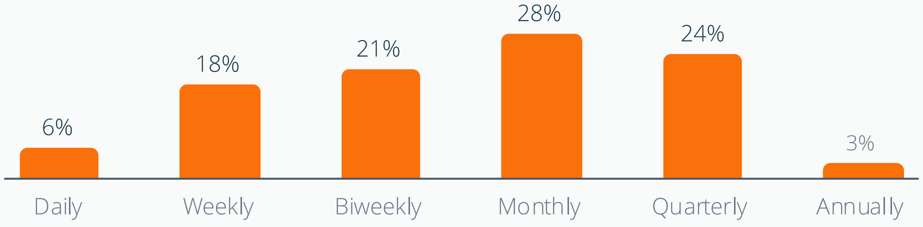
The biggest deterrent to getting a release out on time is *Changing Requirements*. 65% of respondents made this the overwhelmingly top answer to the main roadblock to deadlines. This was the #1 problem in 2019, as well (fig. 20). If we reflect back to the question about which artifacts people are reviewing, remember that *Requirements* reviews are declining. It's possible, and likely, that requirements are changing – but it's also possible that there's no consensus on the requirements that were written, since they're not being reviewed by the larger team.

Regression Bugs, *Waiting on QA*, and *Acceptance Bugs* all nearly tied for second at around 25%. *Waiting on QA* is becoming slightly less of a problem year over year. *Executive Sign-Off* has been less of a roadblock in the last year, down 6%.

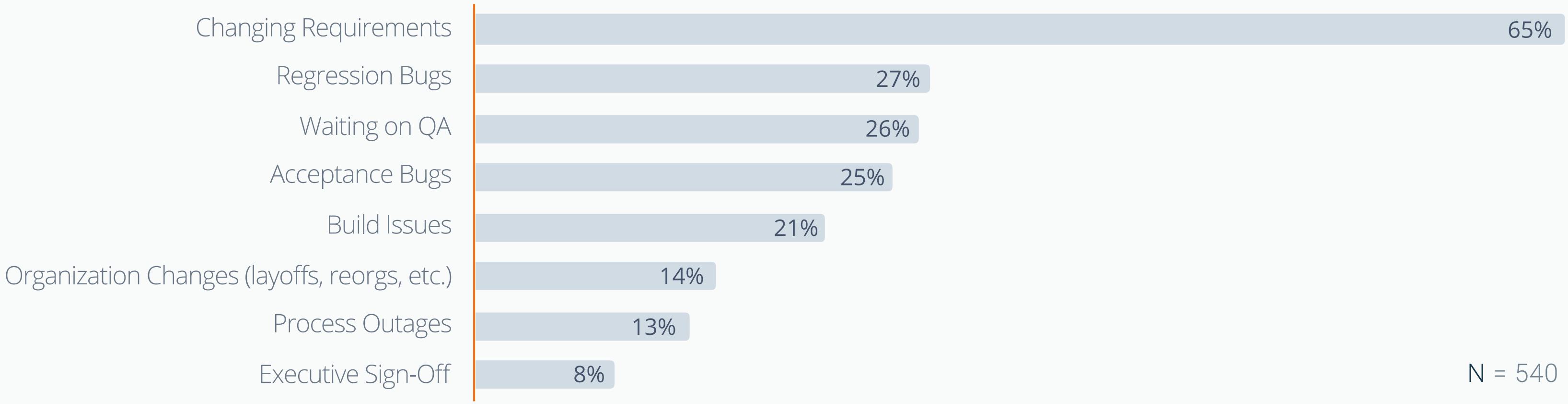
The amount of responses indicating there are no roadblocks that prevent their team from getting a release out on time has increased by 2% since 2019.

How often does your team put out a new release? fig. 19

N = 543



What prevents your team the most from getting a release out on time? *Select all that apply* *fig.20*



A note on saving time

The Navy SEALs train with the idea that *slow is smooth and smooth is fast*. Many software development organizations find themselves going very vast and hoping for smooth results (quality, on-time releases).

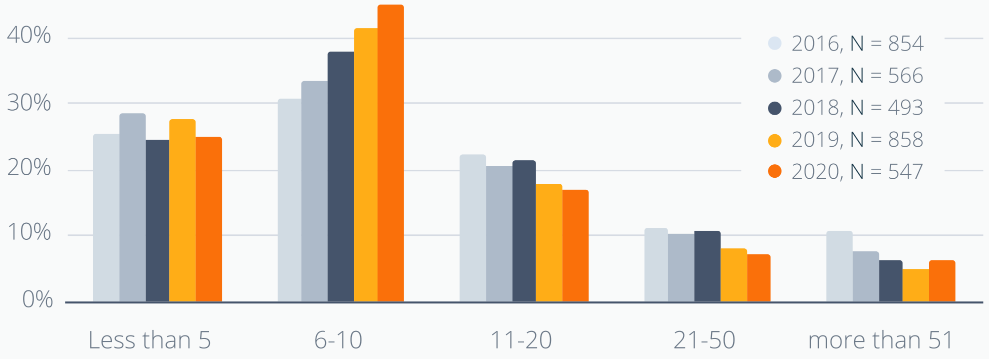
What we know is that taking time to review requirements with our customers and the entire cross-functional team may feel slow, but it ensures that everyone is on the same page. The opportunity for the entire team to review the requirements, and to ask clarifying questions, allows the development process to go more smoothly. If you include the customer's feedback throughout the lifecycle, it will help you stay aligned until the end of the process.

Should the requirements change, which they inevitably do, the team can realign based on the new information and continue to move forward in a 'slow is smooth' approach. This process provides realignment and keeps everything running more smoothly, which is 'fast'.

– Justin Collier, Collaborator Product Manager

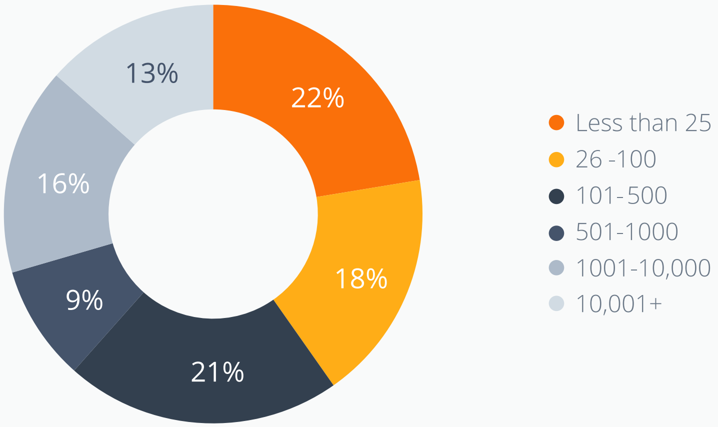
What is the size of the development team you are on?

fig.21



What is the total number of your employees in your company?

fig.22 N = 549



Dev teams increasingly number around 6-10.

When it comes to development team size, 45% of respondents are on a development team of 6-10 people, and 70% are on a team of 10 or fewer *(fig. 21)*. Only 6% are on a development team of over 51 people.

Since 2016, larger teams of 11 or more continue to be on the decline. On the other hand, teams of less than 5, or 6-10, either hold steady or increase. As can be seen, teams of 6-10 have increased by 15% since 2016 and seem to be the size of choice. The latter information makes sense, given the continued transition to Agile methodologies.

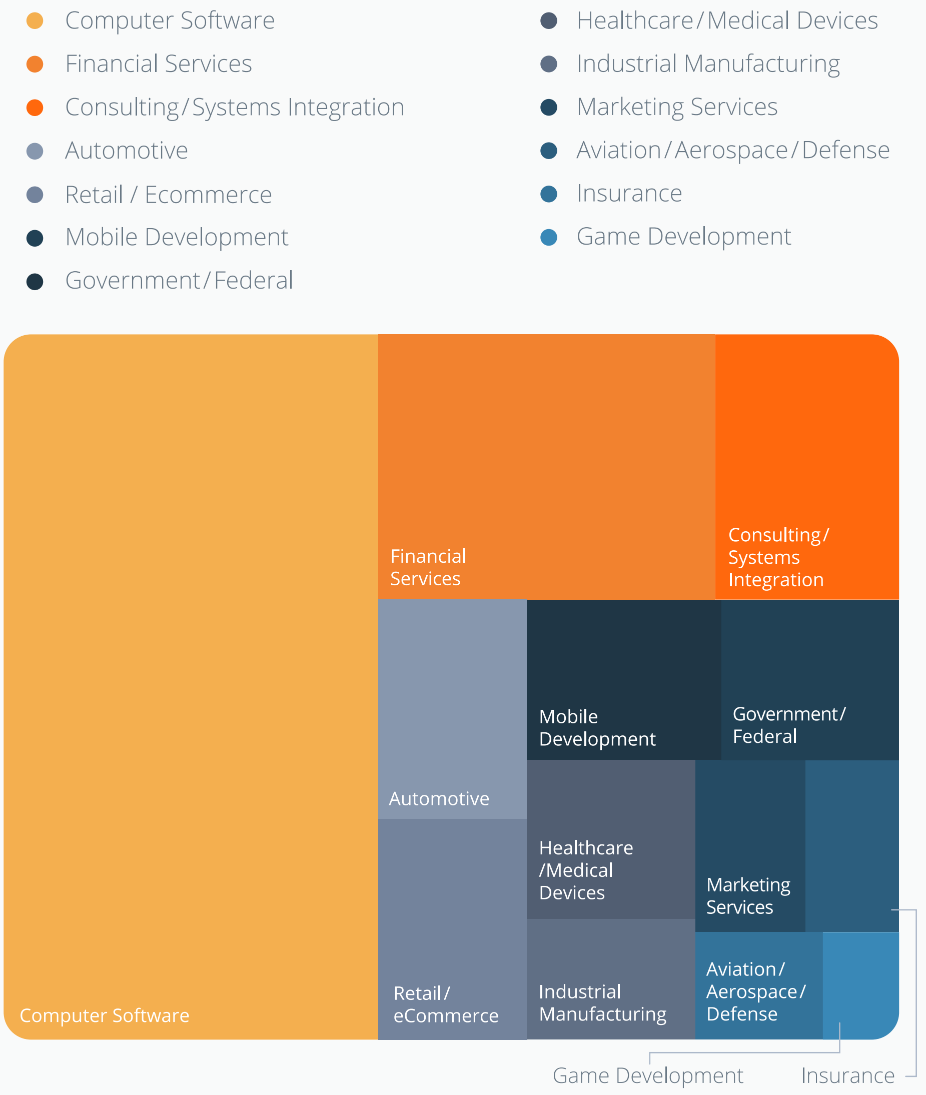
Company size seems to be equally represented in this report.

For total number of employees at the companies of our respondents, about 22% of employees are at a company with fewer than 25 people *(fig. 22)*. 21% work at a company with 101-500 people. Note that respondents from companies of 10001+ employees are down 12% since 2016.

The survey included many respondents from a variety of companies, though 36% of respondents work in the computer software industry, and 12% reside in financial services *(fig. 23)*. We've seen an increase of 12% in additional responses from the financial services industry. The percentage of those in the computer software industry has experienced an increase of 9% from responders.

What industry do you work in? *fig.23*

N = 547



Section 5: Recommendations for Your Team

Code review processes vary from industry to industry and team to team, even within the same organization. This section focuses on the key differences between teams that are either essentially satisfied or dissatisfied with their code review processes. We've removed respondents who were neutral about their code review processes so that the data is easier to digest.

Here are 5 recommendations for teams looking to improve their code review process. Our conclusions are similar to last year, which illustrates how fundamental they are.

1. Daily code reviews are key.

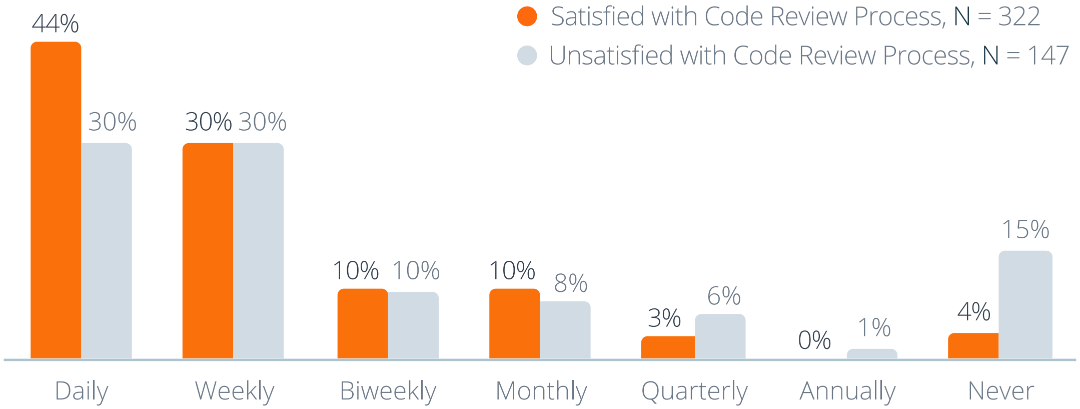
In the chart below, you can see the difference in review frequency between respondents who are satisfied with their code review process and those who are not *(fig. 24)*.

Satisfied teams perform code reviews on a daily basis (44%) compared to teams that are dissatisfied (30%). Last year, the disparity was 53% and 32%, respectively, and in 2018 the disparity was 45% and 22%.

Though some developers feel that code review is disruptive to their productivity, many see it as an opportunity to not only improve the code base but to also reduce bugs, train junior team members, and increase knowledge share across the team.

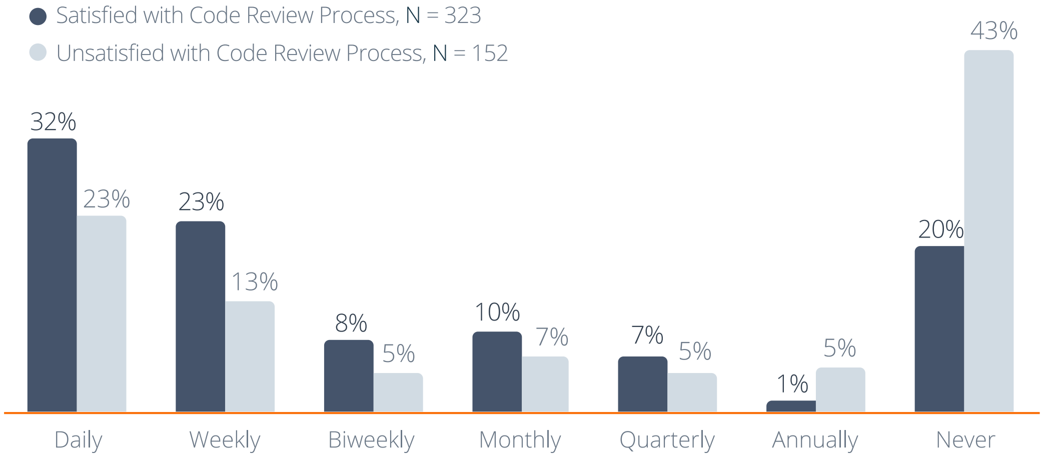
How often do you participate in any kind of code review process?

fig.24



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

fig.25



2. More tool-based reviews, more code satisfaction.

There are several types of approaches to code review. Many teams utilize several of the approaches to meet their needs based on the immediate situation. The approaches include tool-based, ad-hoc (or “over the shoulder”), and meeting-based reviews.

This year’s report reaffirms that taking a tool-based approach makes a major difference on code review satisfaction *(fig. 25)*. Teams that perform tool-based reviews are more likely to be satisfied with their overall code quality. Cohorted by code-review satisfaction, 81% of satisfied respondents are conducting some kind of tool-based reviews. Comparatively, only 58% 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 this case study in which a company reduced their code- and test-review timeline by 70%.

3. Code reviews need clear guidelines.

As mentioned in the "Perceptions of Code Review" section, 56% of respondents answered that "guidelines for how reviews should be performed" have been defined for their team. The chart below shows just how impactful guidelines can be in relation to code-review satisfaction *(fig. 26)*.

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

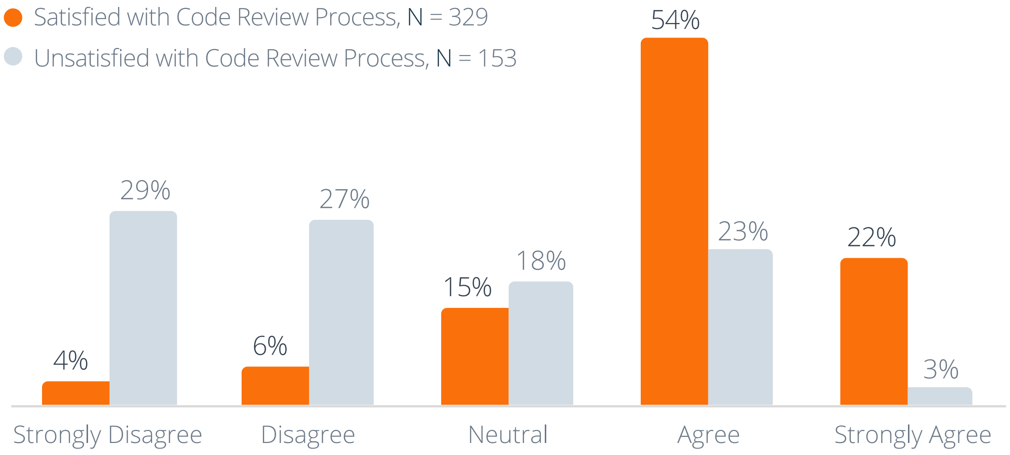
Cohorted by code-review satisfaction, 76% of satisfied respondents have guidelines on how reviews should be performed. Comparatively, only 26% of unsatisfied respondents have guidelines.

The clearer you can set expectations, the more likely your team will produce higher quality software. This applies to both code and document reviews. Additionally, the satisfied code review cohort responded that they review on average 3.6 types of artifacts. The dissatisfied cohort responded that they reviewed, on average, 3 types of artifacts.

How can you ensure that your team is clear on expectations? First, define and assign responsibilities to team members. Second, outline them in a checklist for both code and document reviews. Code-review tools like Collaborator let you build custom checklists in review templates, so participants with different roles and responsibilities can easily see what's expected of them on each project.

My team has guidelines for how reviews should be performed.

fig.26



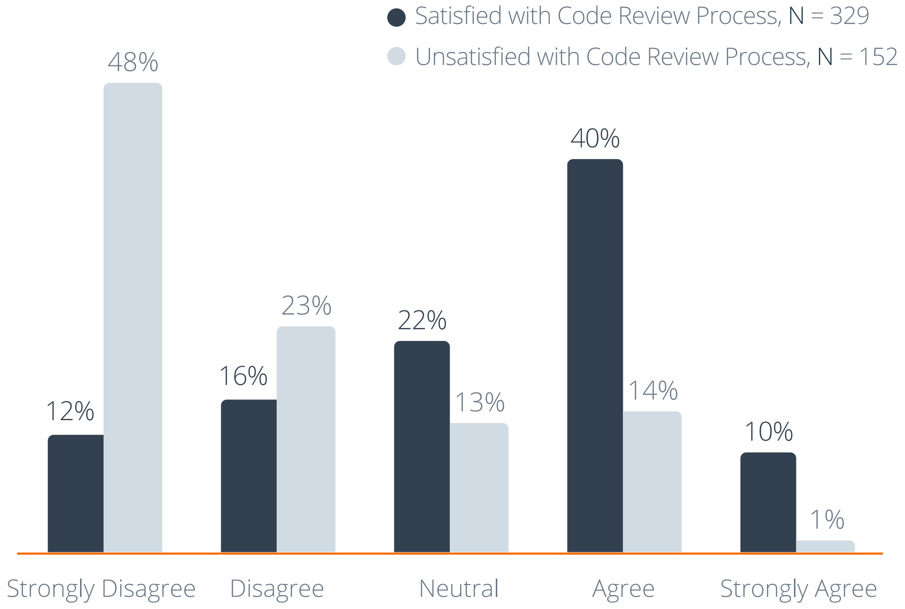
4. Pull reports to get insights on how to improve.

Teams that report on their process are more than 3X as likely to be satisfied with their code reviews (fig. 27).

50% of satisfied reviewers (those who *Agre*e and *Strongly Agree*) regularly pull reports on their process – more than 3 times that of unsatisfied reviewers (15%). It's hard to be satisfied with your code-review process when you can't track its aggregate effectiveness.

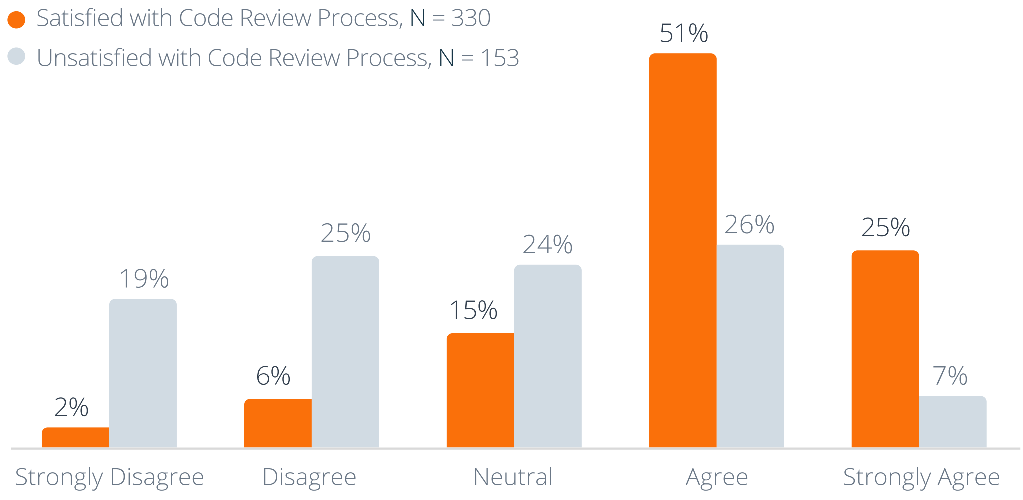
Teams that do review reports and key performance indicators, are more likely to know what to improve when making changes. As an example, tracking the types of defects found, and their severity, can provide insights to aid in process improvement. Adopting a tool that enables you to track key metrics and pull custom reports on peer code reviews is the fastest way to drive meaningful process improvement.

My team regularly pulls reports and metrics on our code review process. fig.27



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

fig.28



5. Code review does double duty in training new people.

When we mentioned 59% of teams use code review as a way to train and onboard new developers, we found those teams are taking the right approach.

This chart shows the breakdown between the two code-review satisfaction cohorts *(fig. 28)*.

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

74% of teams that are satisfied with their code-review process are using it as an onboarding and training tactic – compared to only 33% of the unsatisfied cohort. 81% of all respondents say that they often learn during code reviews, so the pairing is a natural fit.

Teaching and learning are critical to keeping us engaged. Whether you're a senior engineer or new to the team, you have something to offer or can learn something new. Code review can be a vehicle for knowledge sharing as well as daily learning. When your team is learning and collaborating on a daily basis, they'll build trust and improve your code quality at the same time.

FINAL THOUGHT:

Communication and Collective Ownership

Now, more than ever before, we have collectively experienced separation from others. Our desire to be in community is strong, but how can we build community while being so apart? Answer: communication. And collective ownership.

As we reflect on this report, it's easy to see that these two factors play vital roles in our software quality initiatives. If communication improves, our sense of unity improves. It's actually refreshing to engage with peers, and ultimately encouraging when you see the work they're doing.

Even with all that's happened this year, the future of code review looks bright. When we come together behind a common cause and decide to take ownership, our outlook and productivity skyrocket. We begin to help others succeed, and they, in turn, help us. It's amazing what happens when we're all rowing in the same direction. So metaphorically speaking, let SmartBear be your coxswain.

Section 6:

Handing Over the Mic. “What’s *Your* Ideal Code-Review Process?”

One of the ways we can improve our processes is through learning from other people and organizations. For the 2020 report, we asked the following questions: What would your ideal review process look like? What business and team goals would it achieve?

Over 370 gracious individuals responded and provided feedback – which we unfortunately can’t entirely fit into this report. Since our space is so limited, we tried to include varying perspectives. Remember, there’s no “right answer” when it comes to process. However, even though we all have unique needs, as do our businesses, it’s safe to say that we all need process.

Marcelo E. Pujia says:

The ideal review process is the one that ensures having an application where internally (the code), and externally (the functionality), meet the requirements, do not contain errors, and are developed in the most efficient way possible under the standards (which there should be) of the company and the market. The goal is to achieve high quality, low cost, predictable times and customer satisfaction.

Evan says:

Focus on helping more junior devs learn from their mistakes and do better next time. It should be less about squashing individual bugs and more about making the team better.

Felix Lepa says:

Every commit is reviewed by at least one other developer using integrated tooling. Reviews are automatically generated and linked to the system of record (e.g. the story in JIRA). Reviews are completed in a reasonable amount of time (hours/days depending). Review feedback is provided in a constructive manner, and focuses on standards, security, best practices, easy-to-avoid pitfalls, and of course tries to point out mistakes as much as possible.

Team goals include standardized code, mentoring and training of new/junior developers, team coherence, and achieving high customer satisfaction via high code quality.

Business goals include high code quality and customer satisfaction, standards adherence, security and compliance, as well as team and code standardization, and training of developers.

Michele says:

First of all, we have to understand the context and the task goal. So we have to do a process review, regarding the quality standard and well-known practices. We have to control if all the possible tests are implemented. If something may be done well, we have to talk and share our point of view, and decide what to do. When all seems ok, we can set our “green” feedback and allow merge.

+ quality standards
+ sharing knowledge
+ sharing context
+ team building

Ian Willats says:

Ideally the process should be asynchronous, web-based, and tightly-coupled to the software repository, requirements, and test management systems.

The main business goal is to reduce software defects as early as possible, but with the side benefits of sharing knowledge and ensuring consistency and conformance to conventions.

Brian Lowe says:

Junior does the work, senior #1 reviews and adds comments or guidance, senior #2 also reviews and adds comments or guidance, senior #2 merges.

Junior gets educated on what the business considers 'best practice,' as well as technical coding help. Seniors get to understand each others' views and share ideas.

Business benefits by having consistently applied coding standards and best practices.

Vikas Kumar says:

While reviewing, consider these points: 1. Maintain coding standard 2. Methods or file are created as reusable component. 3. Minimize db hit and get limited data required. 4. Better to use library functions until you maintain Big O notation. 5. Consider performance the most important part of your software. 6. It's always helpful to maintain test case for each functionality and keep it updated.

These points will help you to maintain concise, reusable components of codes and, last but not least, less number of bugs introduced in each development cycle.

Mirga Jazbutis says:

All code changes would be reviewed by a developer familiar with the feature being worked on. That is, the reviewer would be able to fully understand the code being changed. Too often, the reviewer is clueless and the review is just a check-off without substance.

Having consistent substantive reviews results in fewer bugs and in fewer forgotten or misinterpreted requirements, leading to a better product that satisfies customer requests. That leads to reduced tech support efforts and increased bandwidth in the development team to add even more new features. And ultimately, it's the addition of new features requested by customers that keep our customers happiest.

Roman Vlasenko says:

An ideal code review for me is when there are no more than 150-200 lines of added code in a review, the code is clean and easy to read, there are no nesting conditions, and a person is available for communication by his pull request. Most importantly, all the goals set in the ticket were completed and tested. If possible, a screencast by code should be recorded. The goals of the company are tested, working, readable and supported code.

Priyanka Sethi says:

Lean, objective, as much automated as possible (e.g. linting, spell-check, merge and test pipelines), checklist, documented guidelines, minimum approval criteria in-place for code merge.

TDD Goals:

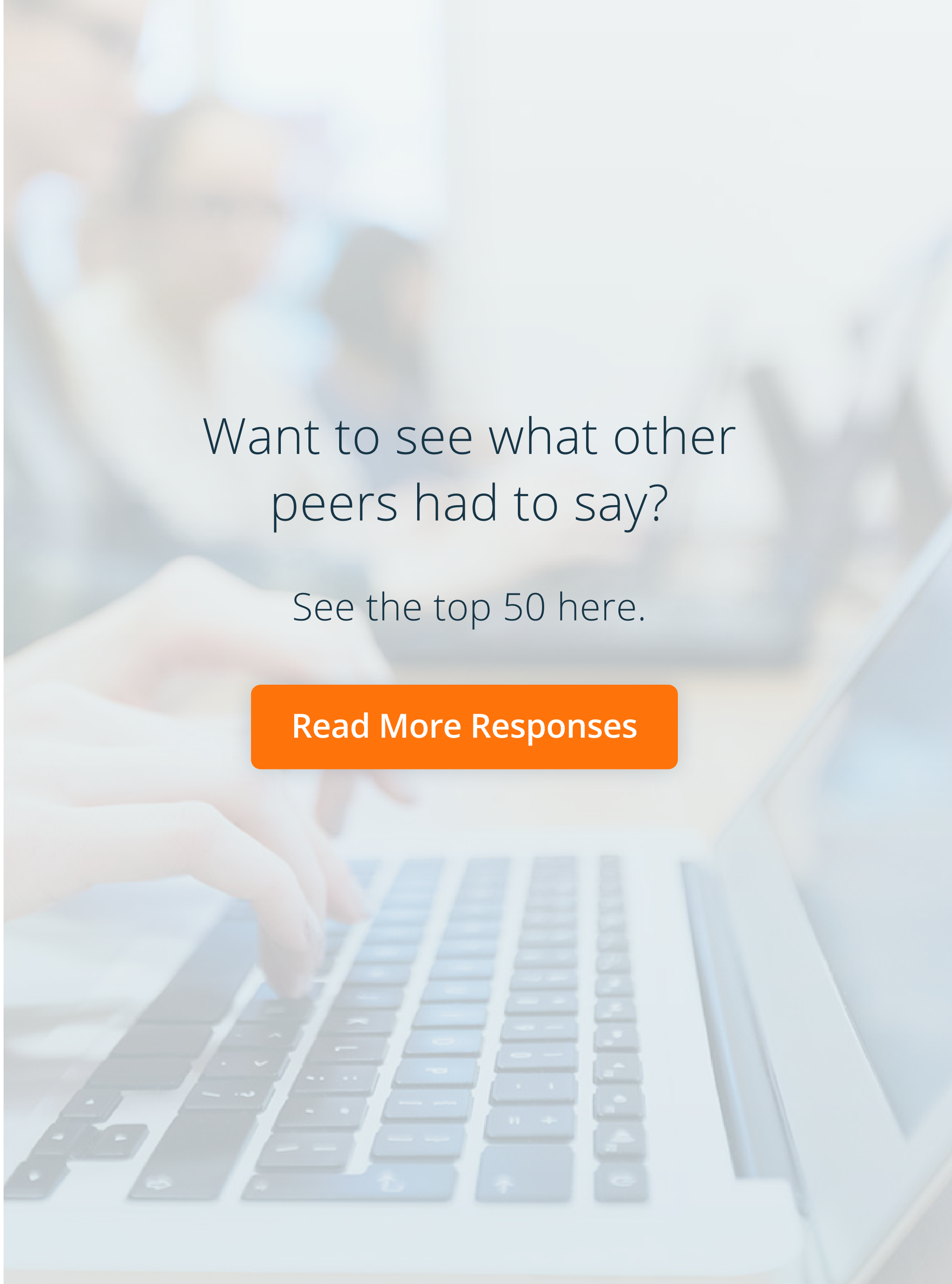
- A certain level of code standards are always maintained
- Reduces probability of merging breaking changes (P0) to the mainline
- New members can learn to write production-ready code quicker than in case of continuous manual knowledge transfer, etc.

Aishani says:

My ideal review process would reward effort and focus on learning. The goal would be to improve people's understanding of what good code looks like.

Meghna Pradhan says:

Review process should have functional as well as non-functional requirements review. It should have peer-to-peer, over-the-shoulder review along with a proper tool-based review. The ideal code review would be done by the development environment where at the time of development itself, there will be guardrails/alerts/warnings shown that will need to be addressed to proceed with the development. At the time of tool-based code review there should be flexibility to override the code review in case of any exceptions, but that should require the next level approval. Before the start of development and whenever any new developer is onboarded, there should be a proper training provided specifying the best practices and coding standards and then the complete step by step process of code review and the criteria to look for. There should be very few exceptions in the code-review process, if we are seeing multiple exceptions, then the review process needs to be updated. Proper review of requirements and simultaneous code review during the development will help meet all business and team goals by eliminating the overhead of peer review.



Want to see what other peers had to say?

See the top 50 here.

Read More Responses

SMARTBEAR

Pro Tools

SwaggerHub

Design, Model, & Share API Definitions

ReadyAPI

Collaborative API Quality Platform

Collaborator

Code, Document & Artifact Review

Open Source Tools

Swagger

Interact With API Resources

SoapUI

Create & Execute API Test Automation

SMARTBEAR