JETBRAINS | Qodana

State of Software Quality Report 2024

Table of Contents

Executive Summary

Welcome to the first issue of the Software Quality Report by JetBrains Qodana!

At JetBrains, we create tools that boost productivity and empower developers to write, test, analyze, and deliver code with confidence. Our mission is to make creating high-quality software enjoyable and consistent.

We conducted a survey involving 808 participants from companies of various sizes – from small startups to large enterprises with over 5,000 employees. Based on insights from the data we collected, this report is designed as a resource to help you navigate and improve your quality practices.

In running this study, our objective was to gain a deeper understanding of best practices for delivering quality software today and figure out how companies of different sizes and product maturity determine the answer to the fundamental question: “When do I know that my product or feature is ready for production?”

Our key discovery is that there’s **a clear shift toward more rigorous practices as products evolve** from launch to maturity, highlighting the increasing complexity of maintaining code quality as companies grow. Larger teams and mature products require stricter controls, which pay off with improved reliability, performance, scalability, and cost savings. Whether you're scaling a startup or managing large enterprise software, these insights will provide valuable guidance to keep your code quality sharp and your releases smooth.

What’s inside

Quality investment trends

Analysis of when and why companies choose to invest more in quality.

Perspectives specific to product maturity and company growth

Insights into how the stage of a project and the company’s size shape its approach to code quality.

Challenges and solutions

A look at the common hurdles in the release and quality assurance process, and some unconventional strategies for overcoming them.

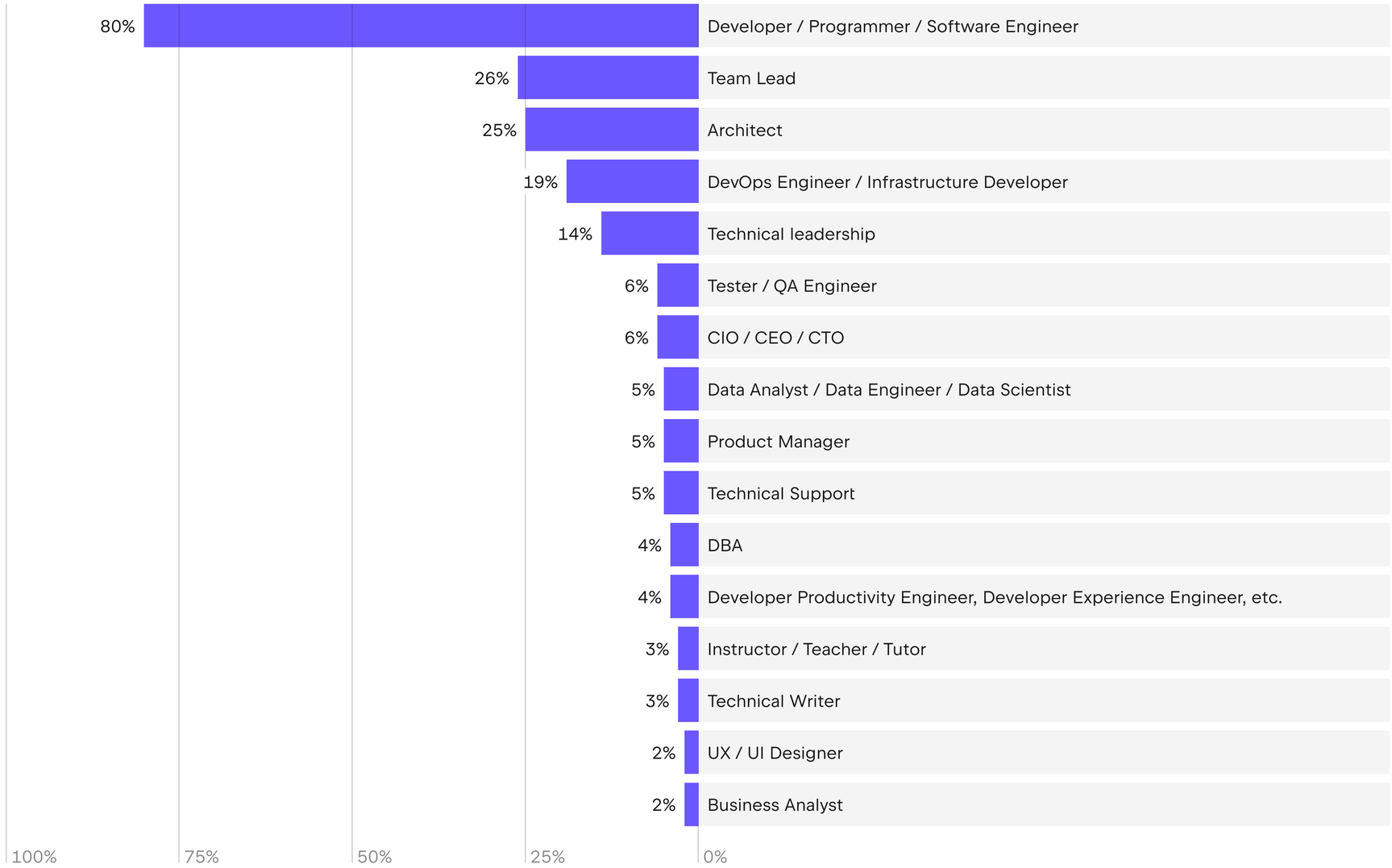
Best and worst practices

A snapshot of how software quality practices are managed, and the key factors influencing decision-making during software releases.

Who We Surveyed

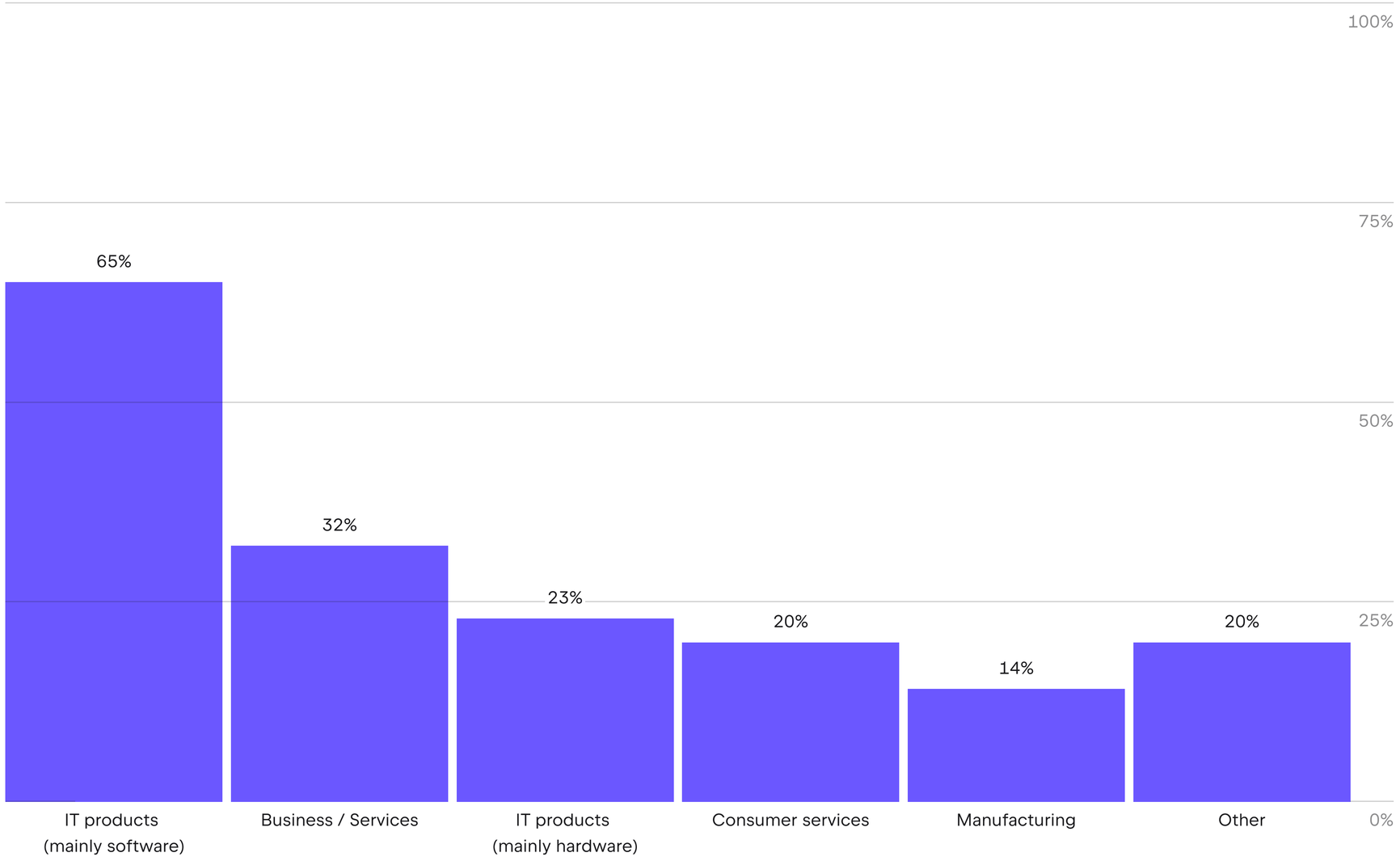
What is your job role?

Survey participants were able to select multiple roles that apply to them.



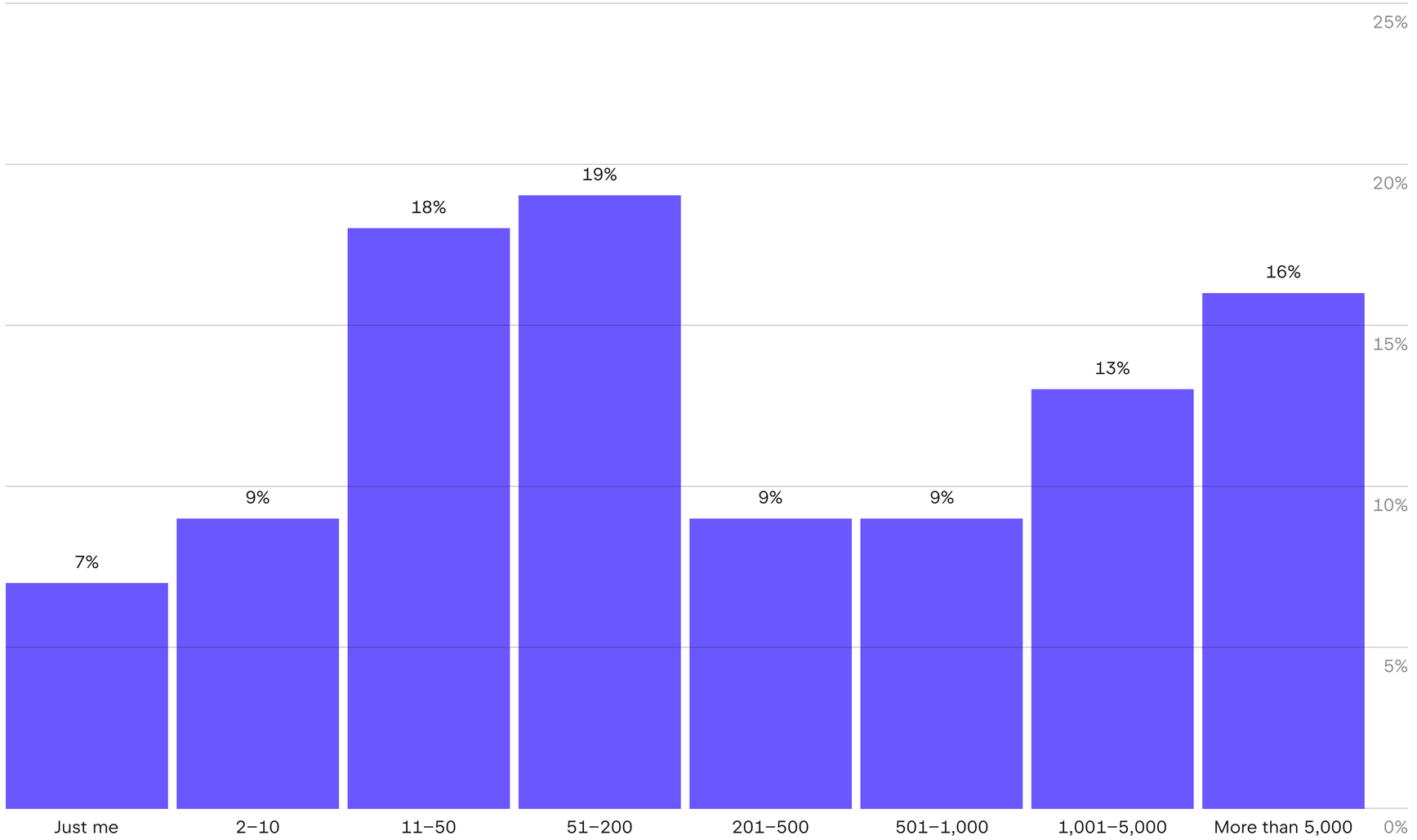
Who We Surveyed

In which of the following sectors is your company primarily active?



Who We Surveyed

How many people work at your company or organization?



Ranking of Practices for Assessing Production Readiness

The top 3 highest-rated practices, as reported by the respondents:

Automated building and deploying (90% rated this as “effective” or “very effective”)

Automated tests (90%)

Automated quality gates for every merge request (80%)

Ranking of Practices for Assessing Production Readiness

Effectiveness of practices to ensure code quality

	Not effective at all 1	2	3	4	Very effective 5
Automated building and deploying	0%	1%	9%	26%	64%
Automated tests	0%	1%	9%	27%	63%
Automated quality gates for every merge request	1%	3%	16%	25%	55%
Use of feature flags for controlled rollouts	0%	4%	18%	37%	40%
Manual testing by a dedicated QA/testing team	1%	5%	17%	39%	38%
Blocking manual (peer) code reviews	2%	7%	21%	38%	31%
Non-blocking manual (peer) code reviews	0%	13%	25%	39%	22%
Manual testing by PM, designers, and engineers	3%	8%	30%	38%	21%
Testing by outsourced teams	4%	10%	33%	33%	20%
Stakeholder approval	5%	16%	34%	28%	18%

Ranking of Practices for Assessing Production Readiness

Key practices that are lacking

We identified several quality controls that many development teams consider desirable but currently lack the resources to implement effectively:

Automated quality gates for every merge request

27%

of teams said they would implement automated quality gates if they had more resources and time. These gates ensure that only top-quality code enters the main branch.

Use of feature flags for controlled rollouts

21%

of respondents expressed a desire to use feature flags. Feature flags allow teams to enable or disable features in production environments, facilitating controlled rollouts and minimizing risks.

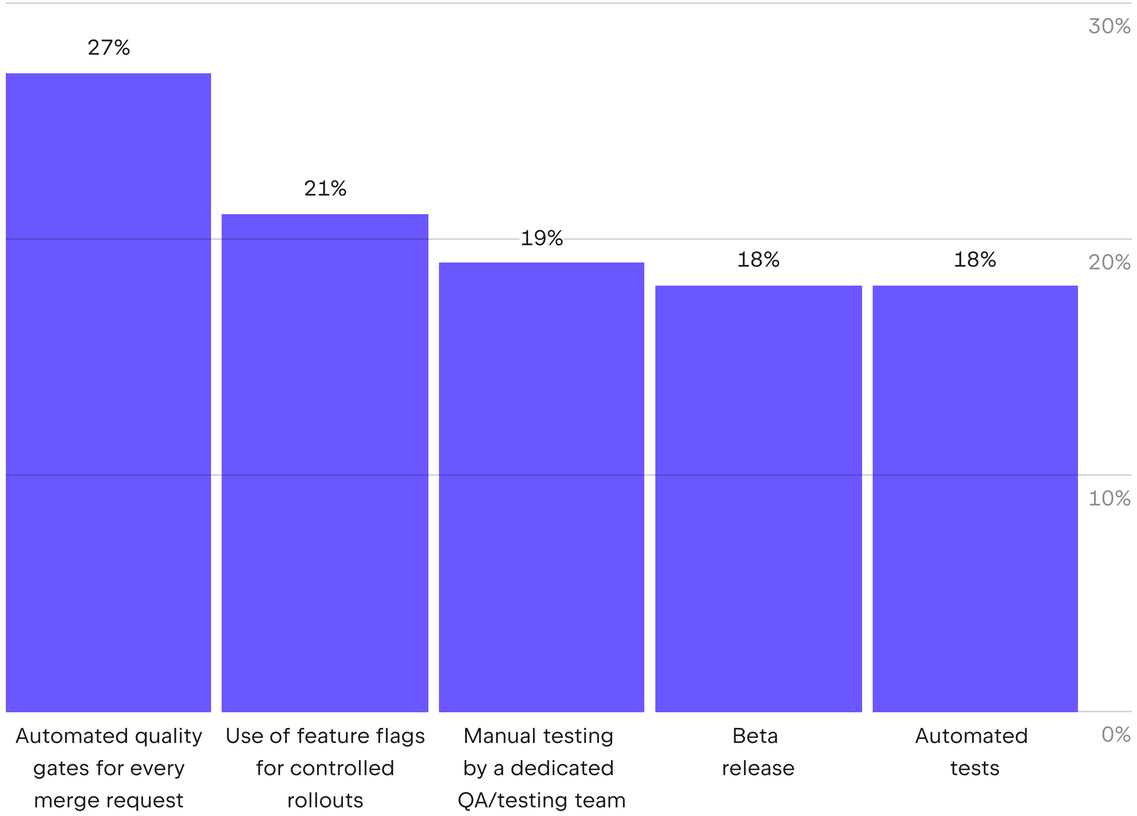
Manual testing by a dedicated QA/testing team

19%

reported this need. Manual testing is essential for identifying issues that automated tests might miss.

Ranking of Practices for Assessing Production Readiness

Top 5 practices that are considered lacking



On the positive side, teams with more mature products are finding ways to address the need for these practices. For more insights, check out our section on how processes evolve as products mature.

Pro tip

Consider implementing automated quality gates with JetBrains Qodana ↗.

Ranking of Practices for Assessing Production Readiness

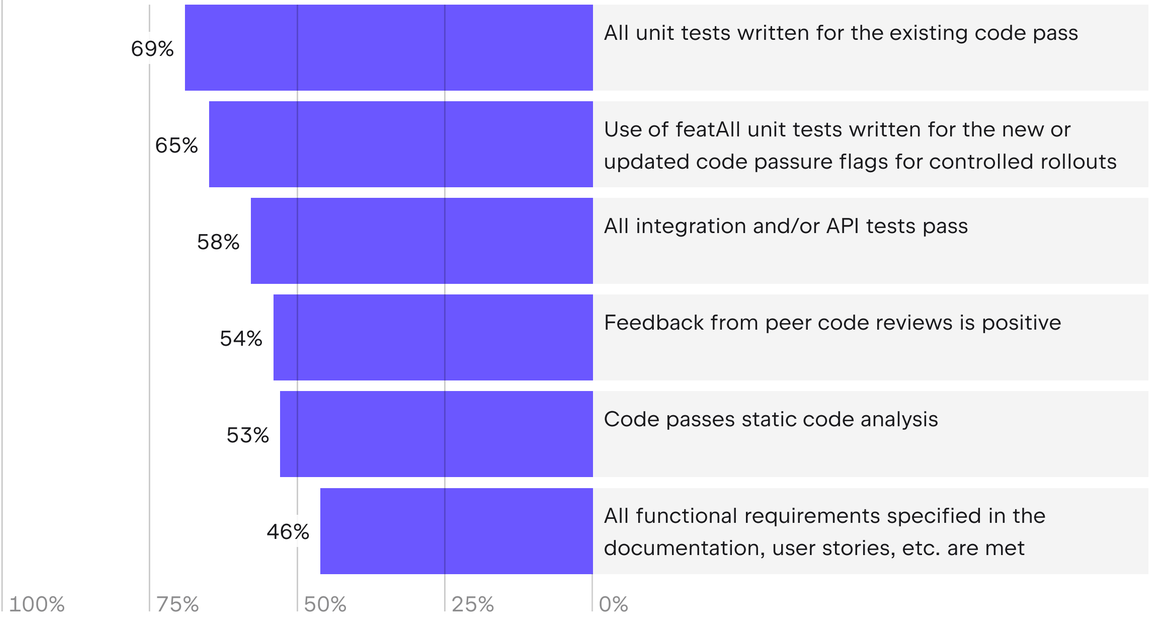
Pre-production quality checks

Typical mandatory checks

Companies typically implement four to six mandatory checks as part of their pre-production quality assurance process (reported by 42%).

6 most common mandatory quality checks

The most common quality checkpoints (at least 45% of respondents said they utilize them) applied to the code before it goes to production:



21%

of respondents have simpler processes involving three or fewer steps, and this practice is more typical for early stages of the product life cycle.

Ranking of Practices for Assessing Production Readiness

Pre-production quality checks

Must-fix issues

At the pre-production stage, data security and privacy issues are considered “must fix” (all issues must be resolved before release) for 60% of respondents. Reliability and stability issues are considered “must fix” only by 37%. These trends are consistent across all company sizes and product maturity levels.

Tolerable issues

Issues are tolerated (resolved only if the team has resources available) mostly in the following categories: Testability (32%), Maintainability (29%), and Usability (22%).

Importance of issues related to code quality

	All issues must be resolved before production	Major issues must be resolved, but not necessarily all issues	Issues are resolved only if we have resources available	We don’t pay attention to it
Data security and privacy	60%	29%	8%	3%
Reliability and stability	37%	53%	9%	2%
Functionality	32%	59%	8%	1%
Compatibility with other products/systems	32%	47%	12%	8%
Testability	19%	42%	32%	6%
Usability	18%	57%	22%	3%
Maintainability	16%	52%	29%	3%
Performance	15%	60%	21%	3%

1% 60%

We also see that mid-size and large enterprises have a relatively complicated release cycle including more steps, compared to smaller companies with under 200 employees. Check out the dedicated section about large companies.

Shift in Traditional Quality Assurance

Manual testing by dedicated internal or outsourced teams remains common, but it's seen as much more resource-consuming and less effective than automated practices.

On the one hand, 39% of respondents use manual testing by a dedicated QA/testing team and find it effective. On the other, the same practice was flagged by 39% of respondents as requiring too much manual effort or workaround solutions – more than any other practice.

In contrast, 61% of teams use automated tests and find them effective in overall software quality assurance.

One respondent highlighted that the requirement to run a certain number of manual tests limits how frequently they can release. This delay leads to more features being added to each release, making the testing process even more complex, especially when it comes to identifying side effects. They described this as a vicious cycle that is difficult to break.

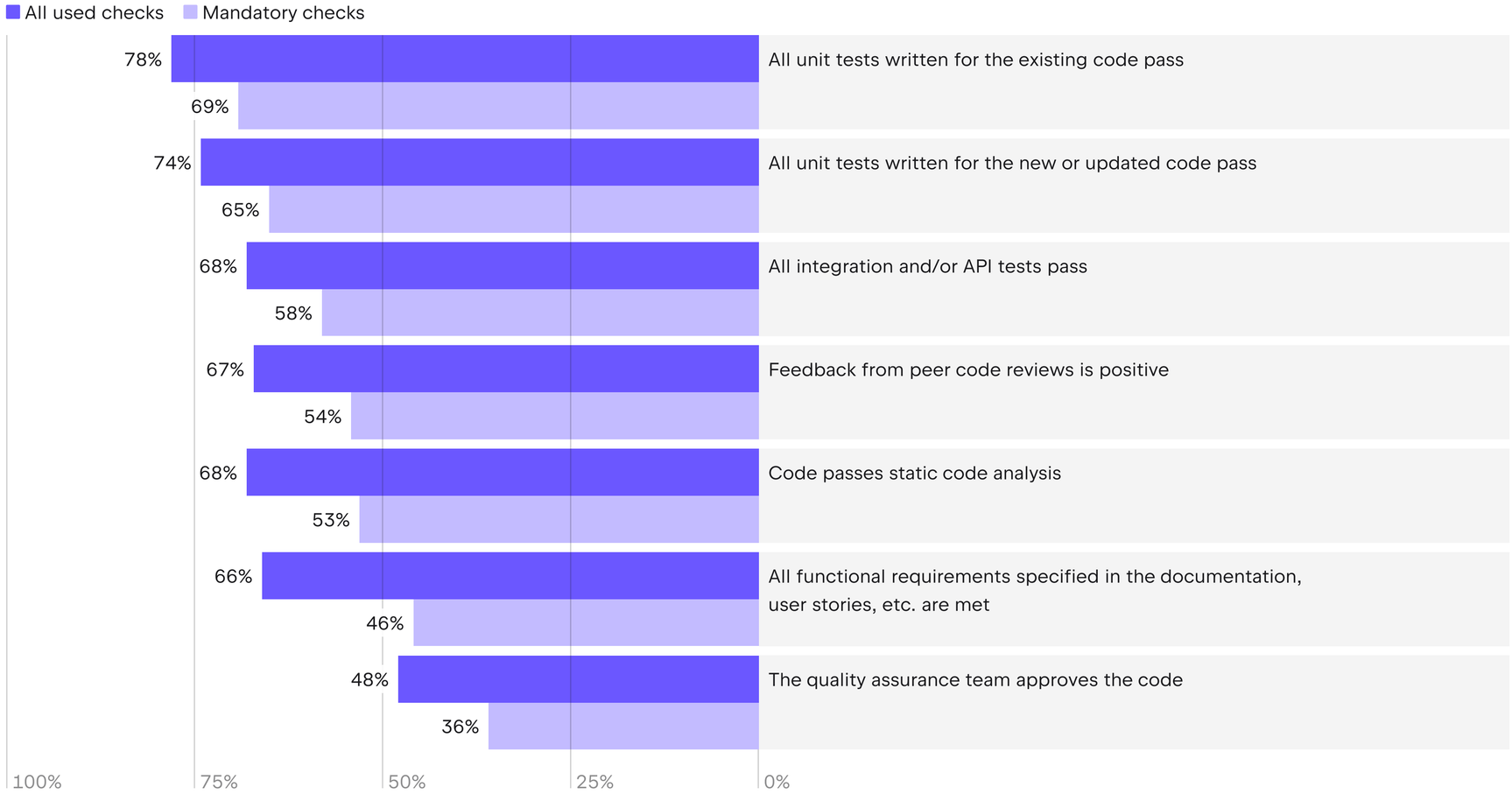
	Have implemented	Consider effective	Consider to be too much effort	Haven't implemented but would like to
Automated tests	68%	61%	15%	18%
Manual testing by a dedicated QA/ testing team	50%	39%	39%	19%
Automated quality gates for every merge request	35%	28%	13%	27%

13% 68%

Unit tests are used by 65%–69% of teams as mandatory checks for code going into production, while 58% also rely on integration tests as a critical step in their process. What's mandatory for releasing code to production is primarily automated checks.

Shift in Traditional Quality Assurance

Top used checks to ensure the code quality



Shift in Traditional Quality Assurance

Given these trends, we anticipate a further shift away from manual quality assurance methods.

Companies are increasingly relying on automated quality gates and controlled rollouts as their products mature. This trend is evident in the growing use of automated quality gates, which rises from **27%** during the **development stage** (before the product has been launched publicly) to **40%** at the **growth stage** (when the product scales after launch), and the adoption of feature flags for controlled rollouts, which jumps from **29%** to **50%.**

In contrast, manual QA is not seeing comparable growth, highlighting the **shift toward more automated and labor-efficient QA practices.**

Find details in the dedicated section about changing practices with product maturity below.

Effect of Product Maturity on Code Quality Approaches

According to one respondent, the development of an actively used software ecosystem is never truly complete. As they noted, the constantly changing business and legal environments mean that if someone believes the software is “ready”, it's time to start considering its eventual sunset.

As products transition from the development and launch stages to growth and maturity, the importance of various code quality practices evolve significantly.

We categorized products into the following stages of maturity:

1

Development

Actively coding and testing initial versions

2

Launch

Recently released publicly

3

Growth

Expanding the user base, adding new features, and scaling business

4

Maturity

Focusing on optimization and efficiency

Effect of Product Maturity on Code Quality Approaches

Our findings reveal that starting from the growth stage, there is a marked increase in the adoption of the following two quality controls that teams lacked at previous stages:

27→40%

Application of automated quality gates for every merge request increases from 27% at the **development stage** to 40% as products enter the **growth stage.**

29→50%

Use of feature flags for controlled rollouts surges from 29% at the **development stage** to 50% as products enter the **growth stage.**

Effect of Product Maturity on Code Quality Approaches

We also observe that as products mature, more practices are adopted overall, and these practices are increasingly perceived as effective.

55→74%

of teams use automated testing at the **development stage** (50% consider it effective), and 74% use it in the **growth stage** (67% consider it effective).

57→72%

of teams use automated build and deploy processes in the **development stage** (52% of teams see value in it), while adoption grows to 72% (and perception of value to 65%) in the **growth stage.**

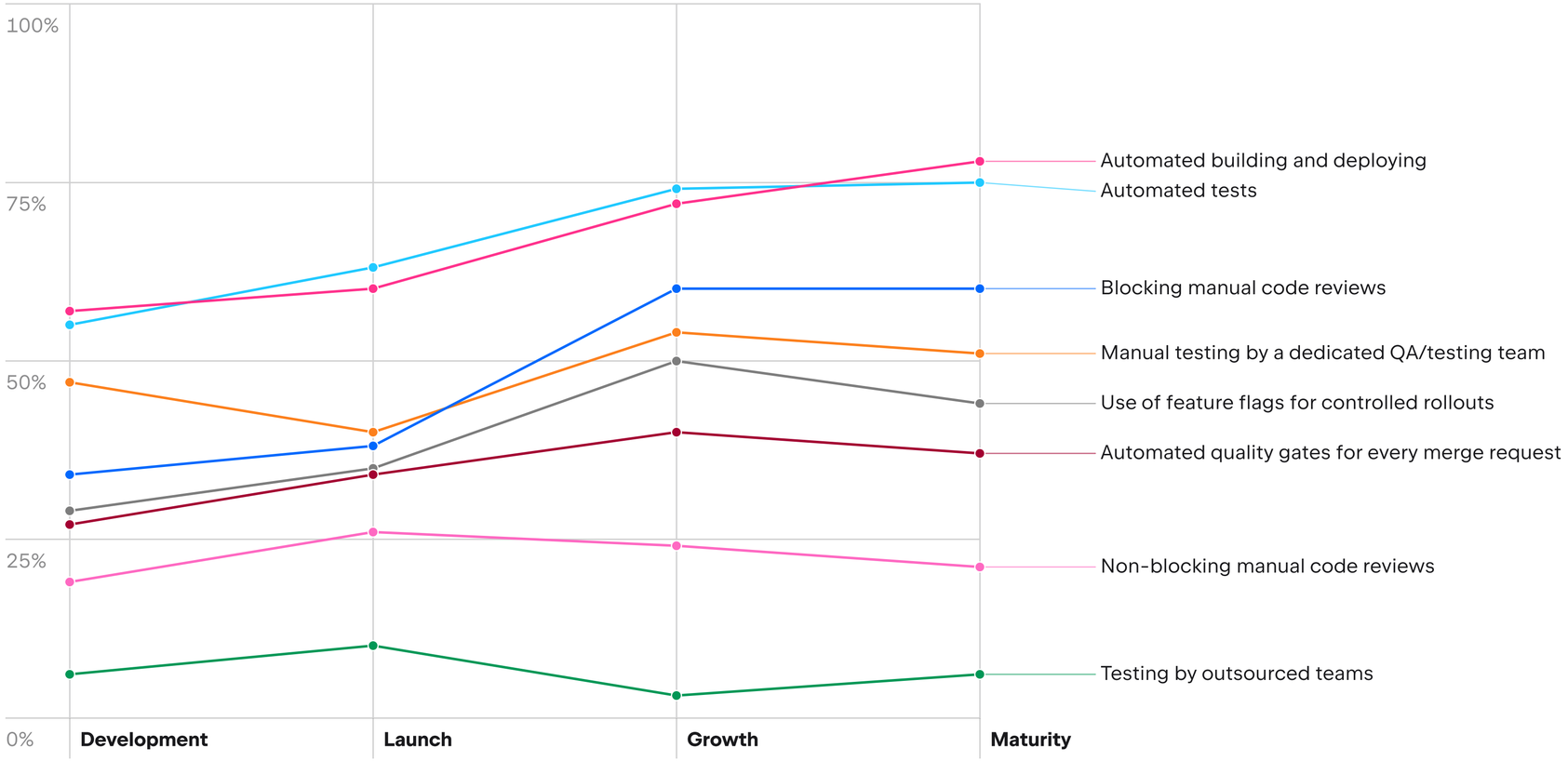
In addition, as products move into the **growth** and **maturity stages,** teams ramp up the use of blocking manual code reviews, jumping from 34%–38% during **development** and **launch** to 60% in the **growth stage.** At the same time, their confidence in the effectiveness of these reviews grows from 24% to 40%. This spike shows that teams feel the need for strict quality checks at every step, but they're leaning heavily on manual peer reviews to get the job done.

Pro tip

This reliance on manual reviews likely points to a gap in automated quality gates. Start investing in automation early to take the pressure off your team and keep your code quality top-notch as your product scales. Consider adopting code analysis and quality automation tools like JetBrains Qodana ↗.

Effect of Product Maturity on Code Quality Approaches

Practices used to ensure software quality at various product stages



These findings highlight the shift toward stricter quality controls as products evolve, likely driven by increased product functionality, complexity, customer base, and reputational risks.

Pro tip

As your product enters the growth stage, invest in CI/CD ↗ and test automation tools ↗, start implementing automated quality gates for every merge request, and use feature flags for controlled rollouts.

Scale Matters: Changes in Release Cycles as Companies Grow

As companies expand from small teams (under 50 employees) to large enterprises (more than 1,000 employees), the complexity and rigor of their release cycles increase significantly and the following steps become more prevalent:

1

Code reviews

While **69%** of the smallest companies conduct peer reviews, this number jumps to **88%** in companies with over 1,000 employees.

2

Unit testing

The use of unit tests is much higher in larger organizations. **84%** of large enterprises implement unit testing, whereas only **62%** of small companies do.

3

Integration testing

Similarly, integration testing becomes more common as company size increases, from being used by **56%** of the smallest companies to **80%** of the largest.

4

Quality assurance (QA) testing

QA testing is more emphasized in larger companies, with **72%** of large enterprises conducting it, compared to **52%** of small companies.

5

Logs monitoring

Larger companies are more likely to actively monitor product logs (**70%** vs. **55%**).

Scale Matters: Changes in Release Cycles as Companies Grow

Release cycle steps by company size

Number of employees:	≤50	51–200	201–1,000	>1000
Code review: conducting peer reviews of the code	69%	81%	83%	88%
Unit testing: verifying that individual units work as intended by themselves	62%	70%	78%	84%
Integration testing: verifying that components work as intended with each other	56%	61%	75%	80%
Quality assurance testing: testing the product in a staging environment to ensure it meets requirements and quality standards	52%	62%	72%	72%
Monitoring: actively monitoring the product logs for issues	55%	49%	63%	70%

Scale Matters: Changes in Release Cycles as Companies Grow

The specific quality controls that see increased use as the company grows include:

Automated quality gates

Automated building and deploying

Blocking manual code reviews

Manual testing

Feature flags

Scale Matters: Changes in Release Cycles as Companies Grow

Practices used to ensure software quality across company sizes

Number of employees:	≤50	51–200	201–1,000	>1000
Automated tests	60%	67%	69%	78%
Automated building and deploying	59%	63%	69%	77%
Manual testing by a dedicated QA/testing team	39%	51%	59%	59%
Blocking manual code reviews	42%	47%	57%	58%
Use of feature flags for controlled rollouts	32%	39%	48%	49%
Automated quality gates for every merge request	25%	29%	45%	43%

Pro tip

As your company grows, invest in automation early to tackle scaling challenges. Our survey shows that the larger a company is, the more often manual testing is used, even though it is among the least effective and most effort-intensive practices. Instead of adding more people to scale manual testing, focusing on scalable solutions like automated building, deploying, and quality gates may be more cost-effective. Start integrating quality checks now, like small, consistent code commits, to build a cleaner and more resilient product. Prioritizing automation will create a more efficient, scalable, and effective approach to maintaining code quality.

Common Foes of Quality

Even if there's company-wide consensus on implementing software quality checks, tech leadership encounters two common problems:

1

Technical challenges (52%)

Conflicting processes, manual workflows, and usability issues can make it difficult to implement consistent quality checks across the board.

Respondents experiencing technical challenges said they lack proven methods to automate quality measurements and tracking, which could have helped them prepare their teams for the shift toward more quality-oriented approaches.

2

Organizational and process issues (45%)

Low buy-in from business management and misaligned priorities can create friction, slowing down or complicating the implementation of quality practices.

One of our respondents said software developed at their company is often being evaluated using metrics like frequency of deployments, Scrum Sprint Burndown Charts, etc. This leads to teams focusing on metrics themselves, sacrificing product quality in favor of "good-looking numbers".

This sentiment was shared by many other respondents, who admitted they, too, often sacrifice code quality to ship features or fix bugs fast.

We've collected some non-obvious ways to overcome these challenges and turn software quality controls into an iterative continuous process. Read about them in the Recommendations section.

The Return on Code Quality Investment Is Real

As was shown above, larger companies with more developers and those with mature products often have complex release cycles because the risks and the costs of low-quality code are too high to ignore.

Here's How Investing in Quality Pays Off

Higher scalability potential

Well-written code scales easily as the business grows, handling increased loads and new requirements without major overhauls.

Saving time on redoing work

Investing in code quality upfront reduces costs by minimizing modifications and emergency fixes.

Easier onboarding and retention of developers

High-quality, consistent code not only shortens onboarding time for new developers but also makes the work environment more satisfying, encouraging developers to stay longer with the team.

Improved reliability and performance

Quality code leads to fewer bugs, crashes, and performance issues, making software more reliable and boosting customer satisfaction, retention, and word-of-mouth.

> "really paid off over the years."

When answering our open-ended question about additional insights on the topic of quality, one respondent said that putting software quality first and investing in code readability and correctness, as well as lots of unit and integration testing, has "really paid off over the years." As a result, developers in their company now spend most of their time developing new features, and not chasing bugs.

Bonus Section: Products Associated With High Quality

We asked an open-ended, unaided question to learn what products our respondents perceive as high quality, and here's what we gathered:

JetBrains IDEs are chosen for their:

- Reliability
- Feature set
- UX
- Customer support
- Compatibility
- Integration with other technologies

Google services are chosen for their:

- Reliability
- Performance
- Intuitive interface
- Updates
- Wide usage

Apple products are chosen for their:

- Reliability
- Performance
- Simplicity
- Interface and UX
- Updates and fixes

Recommendations

Focus on new and changed code only

To make the adoption of new practices manageable, focus on newly written code or the code you're changing – don't try to cover the whole codebase with tests or static analysis in one go. Implement one thing at a time to steadily improve the codebase without exerting unnecessary pressure.

Implement a Quality Champion program

Designate Quality Champions within each team or department. These teammates will advocate for best practices and ensure that quality standards are being maintained. By bridging the gap between technical teams and management, Quality Champions will help spread a culture of quality assurance across the entire company.

Incrementally automate high-impact steps

Begin automating the most impactful quality measures, such as quality gates in your CI/CD pipeline or the longest manual tests. Start with automating a couple of routine code quality checks or tests, and then gradually introduce more and more automation. This will let you adopt code quality practices without overwhelming the team.

Choose tools that are readily available to developers

Ensure that your code analysis tools are configured to deliver prompt and actionable feedback directly to developers. For instance, set up static analysis tools to immediately alert developers of issues within their IDE, providing clear instructions and easy-to-apply solutions.

Pilot new practices in low-risk areas

When introducing new practices or tools, start with low-risk areas of your product or process. Refine approaches and demonstrate their value before scaling them across the organization, minimizing resistance to change.

Time-box quality gates

To balance the need for thorough quality checks with the pressure to meet deadlines, introduce time-boxed quality gates. This could involve setting strict time limits for peer reviews or automated tests, ensuring that they do not delay the release process.

Reinforce positive behavior

Celebrate small wins when the team successfully integrates a new tool or practice. Provide immediate feedback to encourage continuous improvement, and foster a supportive environment.

Read more: How to put together a team of quality-focused developers ↗

Final Thoughts

We hope that you're leaving with more than just insights – that you've got a roadmap, a compass, and maybe even a few new tools for your software quality toolkit. Keeping your code in top shape isn't about conquering the whole mountain in one leap; it's about taking smart, steady steps up the trail.

Whether you're a small startup or a large enterprise, the journey to high-quality software is like learning to cook. You don't need to cook every dish perfectly on the first try. Start with the freshest ingredients (your new code), use the right tools (automated testing, quality gates), and adjust your recipe as you go.

Most importantly, don't forget to enjoy the process – celebrate those little wins along the way, and before you know it, you'll be serving up a product that's nothing short of a five-star experience.

Keep pushing the envelope, keep iterating, and happy coding!

JETBRAINS | Qodana

Make it happen.
With code.